The Dissertation Committee for Gary W. Kinney Jr.

certifies that this is the approved version of the following dissertation:

A Group Theoretic Approach to Metaheuristic Local Search for Partitioning Problems

Committee:

J. Wesley Barnes, Supervisor

Bruce Colletti

John Hasenbein

Erhan Kutanoglu

Elmira Popova

A Group Theoretic Approach to Metaheuristic Local Search for Partitioning Problems

by

Gary W. Kinney Jr., B.G.S., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May, 2005

Dedication

To my wife Mary and my boys, Nathan and Ryan

All truths are easy to understand once they are discovered;

the point is to discover them.

- Galileo Galilei

Italian astronomer & physicist (1564 – 1642)

Acknowledgments

I would like to thank the faculty and students in the Operations Research department for sharing with me their time, knowledge, and excitement for OR. I will carry with me many fond memories of my time with you at UT.

I would like to thank Dr. John Hasenbein, Dr. Erhan Kutanoglu, and Dr. Elmira Popova for agreeing to be a part of this research and subjecting themselves to yet another group theoretic adventure.

I would especially like to thank Dr. Bruce Colletti for planting the seeds and the passion for group theoretic tabu search. His excitement over even the smallest discovery was contagious and kept me enthusiastic and hopeful even when I thought all was lost.

I would like to thank my committee chair Dr. J. Wesley Barnes for inspiring me to come to UT to work under his guidance and giving me the freedom to choose my own path. His counsel during this research was invaluable and I certainly would not have been able to accomplish it without him.

Finally and most importantly, I would like to thank my wonderful wife Mary who convinced me to chase after my PhD. She has been by my side through this entire program keeping me grounded when I got too high and picking me up when I got too low. She has also given me two magnificent boys who care nothing about group theory and tabu search and just want their daddy to sit and play with them: a distraction I often needed and always cherished.

A Group Theoretic Approach to Metaheuristic Local Search for Partitioning Problems

Publication No._____

Gary W. Kinney Jr, Ph.D.

The University of Texas at Austin, 2005

Supervisor: J. Wesley Barnes

Recent work has demonstrated the power of combining group theory with metaheuristic search methodologies to solve discrete optimization problems. Group theory provides tools to characterize the underlying structures in move neighborhoods, solution representations and solution landscapes. Exploiting these structures with group theoretic techniques produces highly effective and efficient search algorithms.

Using group theory we develop a methodology for partitioning the solution space into orbits. The partitioning is performed by clustering the variables based on the reduced costs of the LP relaxation creating “good” and “bad” orbits. We are able

to calculate the size of each orbit and place upper and lower bounds on the solutions contained within. The search efforts can then be directed on the "good" orbits.

Based on these ideas, we develop a Group Theoretic Tabu Search (GTTS) algorithm for solving the unicost Set Covering Problem (SCP), GTTS-USCP. We tested our algorithm on 65 benchmark problems and compared the results against the previous best known and solutions obtained by CPLEX 9.0. GTTS-USCP discovered 46 new best known solutions. GTTS-USCP converged significantly faster than CPLEX for all problem sets.

We explore the general integer linear program (ILP) by way to the group minimization problem (GMP). By examining the local search in terms of the GMP, we gain insights that will help us solve the ILP. We describe the local search for the corner polyhedron in the space of the non-basic variables. Integer points in the corner polyhedron that produce an all integer basis form a sub-lattice. We develop identity move neighborhoods that allow the local search to traverse this sub-lattice. We also develop bound strengthening of the non-basic variables based on reduced costs. These bounds effectively shrink the corner polyhedra reducing the size of the solution space we must search.

Based on this research, we develop a GTTS algorithm for solving the GMP, GTTS-GMP. Since the GMP can be formed from any ILP, this algorithm solves the general ILP. The algorithm performs well on a diverse set of benchmark problems.

Table of Contents

Notation

Acronyms

AR(1)	1st Order Auto Regressive Time Series
FGMP	Factor Group Minimization Problem
GMP	Group Minimization Problem
GTTS	Group Theoretic Tabu Search
GTTS-GMP	GTTS algorithm for the group minimization problem
GTTS-USCP	GTTS algorithm for the unicost set covering problem
ILP	Integer Linear Program
MDKP	Multi-Dimensional Knapsack Problem
m-STSP	Symmetric Traveling Salesperson Problem (multiple)
RTS	Reactive Tabu Search
SCP	Set Covering Problem
SOS	Special Ordered Sets
TS	Tabu Search
TSP	Traveling Salesperson Problem
WSCP	Weighted Set Covering Problem

Symbols

S_n	symmetric group on n letters
$\langle g \rangle$	group generated by group element g
g^n	n-fold product of group element g
e	group identity
g^h	conjugate of g by h
$N \lhd G$	N is a normal subgroup of G
${}_G T$	group action of G on T
$G(D)$	group of fractions of the form $k/D,\ k, D \in \mathbb{Z},\ 0 \le k < D$

$Z(D)$	cyclic group of size D consisting of elements $\{0,1,\ldots,D\text{-}1\}$
$M(\boldsymbol{I})$	group of integer vectors of size m
$M(\boldsymbol{B})$	subgroup of all linear-integer combinations of columns in the optimal LP basis
$M(\boldsymbol{I})/M(\boldsymbol{B})$	factor group $M(\boldsymbol{I})$ modulo $M(\boldsymbol{B})$
P	feasible polyhedron of an LP
CP	corner polyhedron
$\boldsymbol{O}^p$	characterization matrix for orbit p
$z_{\mathrm{UB}}{}^k$	upper bound on cost/benefit of variables in cluster k
$\boldsymbol{A}$	size m x n incidence matrix for an LP or ILP
$\boldsymbol{x}$	size n column vector of decision variables
$\boldsymbol{c}$	size n row vector objective function costs
$\boldsymbol{b}$	size m column vector of right hand side values
$\boldsymbol{s}$	size m row vector of slack/surplus variables
$\boldsymbol{B}$	size m x m matrix composed of the basic variable columns
$\boldsymbol{N}$	size m x n matrix composed of the non-basic variable columns
$\boldsymbol{x}_N$	size n column vector of non-basic variables from optimal LP solution
$\boldsymbol{x}_B$	size m column vector of basic variables from optimal LP solution
$\bar{\boldsymbol{c}}_N$	size n row vector of reduced costs for the non-basic variables
$\boldsymbol{x}^0$	size n column vector starting solution for a search algorithm

$\boldsymbol{x}^*_{\mathrm{LP}}$	size n column vector containing an optimal solution to an LP
z^*_{LP}	value of an optimal solution to an LP
z_{LB}	lower bound on ILP solution value
z_{UB}	upper bound on ILP solution value
z	value of current solution to an LP or ILP
x_{NUj}	j^{th} non-basic variable at upper bound
x_{NLj}	j^{th} non-basic variable at lower bound
$\bar{c}_{NUj}$	reduced cost for j^{th} non-basic variable at upper bound
$\bar{c}_{NLj}$	reduced cost for j^{th} non-basic variable at lower bound
$\Omega\left(\boldsymbol{O}^p\right)$	orbit hash function used in GTTS-USCP
$\varphi\left(\boldsymbol{x}\right)$	solution hash function used in GTTS-USCP and GTTS-GMP
$\Delta\varphi\left(\boldsymbol{x}\right)$	change in solution hash function as a result of the move
$\boldsymbol{\alpha}_j$	column vector containing the fractional parts of $\boldsymbol{B}^{-1}\boldsymbol{N}_j$
$\boldsymbol{\alpha}_0$	column vector containing the fractional parts of $\boldsymbol{B}^{-1}\boldsymbol{b}$
$H(\boldsymbol{\alpha})$	subgroup generated by the columns of the GMP
D	absolute value of the determinant of the basis $\boldsymbol{B}$
$K(\boldsymbol{B})$	cone generated by the columns of $\boldsymbol{B}$
$\left\|\boldsymbol{N}_j\right\|$	L1 norm of vector $\boldsymbol{N}j$
$\boldsymbol{X}$	collection of decision variables excluding slacks
$\boldsymbol{X}_{\boldsymbol{N}}$	collection of non-basic variables
$CP^{\boldsymbol{X}}$	corner polyhedron in $\boldsymbol{X}$ space
$CP^{\boldsymbol{X}_N}$	corner polyhedron in $\boldsymbol{X}_N$ space
K	cone formed by binding constraints at $\boldsymbol{x}^*_{\mathrm{LP}}$

$P^{\boldsymbol{x}_N}$	polyhedron formed by relaxing the GMP constraints
$\nabla \boldsymbol{x}$	change in decision variables for identity move j
$\nabla \boldsymbol{s}$	change in slack/surplus variables for identity move j
$\boldsymbol{d}^j$	size n + m column vector detailing $\nabla \boldsymbol{x}$ and $\nabla \boldsymbol{s}$ for identity move j
p_j	multiplier for identity move j for a compound move
k	bound on sum of absolute value of p_j's
l	lower bound on p_j's
u	upper bound on p_j's
$N(k, l, u)$	identity move neighborhood based on k, l, and u
$\hat{\boldsymbol{x}}_{\text{GMP}}$	size n column vector containing the current best solution found for the GMP
$\hat{z}_{\text{GMP}}$	value of the current best solution found to the GMP
z_{GMP}	value of the current solution for the GMP
lb_j	lower bound for decision variable j
ub_j	upper bound for decision variable j

Chapter 1 - Introduction

Recent work has demonstrated the power of combining group theory with metaheuristic search methodologies to solve discrete optimization problems. Group theory provides tools to characterize the underlying structures in move neighborhoods, solution representations and solution landscapes. Exploiting these structures with group theoretic techniques produces highly effective and efficient search algorithms.

Discrete optimization problems may be divided into three distinct groups: partitioning, ordering and partitioning-and-ordering problems. Partitioning problems such as set covering, knapsack and min-cut network flow problems have no ordering context and require only that the solution variables be placed into mutually exclusive sets. Ordering problems such as single-agent traveling salesman, single-machine job shop scheduling and single-vehicle routing problems require that a permutation of the solution variables be stipulated. Partitioning-and-ordering problems such as multiple-agent traveling salesmen, multiple-machine job shop scheduling and multiple-vehicle routing problems require that the solution variables be partitioned and ordered within each partition.

Since previous group theoretic metaheuristic research has focused on either ordering or partitioning-and-ordering problems, only the symmetric group on n letters, S_n, the group of permutations of n distinct objects, has been employed. S_n is inappropriate for strict partitioning problems. An appropriate direction for current

research is to investigate whether group theory can improve the performance of metaheuristic search methodologies for partitioning problems.

In this dissertation, the use of group theory for the characterization of move neighborhoods, solution representation and landscape structures for partitioning problems is discussed. Search methodologies that exploit these structures appear to achieve better solutions in less time than competing approaches.

In addition, Gomory's long neglected group minimization problem (GMP) is addressed. The GMP seeks to construct the optimal solution to an integer linear problem (ILP) from the optimal solution to its linear relaxation. In the 1970s, there were several attempts to solve the GMP with dynamic programming. This research reexamines the GMP from a metaheuristic perspective and describes local search neighborhoods for solving the general ILP in this context.

We will now overview the remaining chapters in this dissertation. Chapter 2 provides a brief overview of group theory and a review of previous operations research literature involving group theory. Chapter 3 considers the general partitioning problem and presents group theoretic strategies for partitioning the solution space to enhance the exploration of that space. Chapter 4 provides a highly effective and efficient reactive tabu search (RTS) implementation of one of these strategies for the unicost set covering problem (SCP).

Chapter 5 describes move neighborhoods for the general IP in terms of group theory and the GMP. Since any ILP can be transformed into a GMP, these ideas can be applied within most metaheuristics to create a search algorithm that solves the

general ILP. Chapter 6 presents a powerful RTS algorithm for the GMP employing these neighborhoods and ideas.

Finally, Chapter 7 provides conclusions from this research as well as directions for further research.

Chapter 2 - Literature Review

This chapter presents a synopsis of group theory and the previous research relevant to the goals of this dissertation. Section 2.1 presents a brief overview of group theory. A more comprehensive treatment can be found in countless abstract algebra texts such as Fraleigh (1976) or Herstein (1975). Colletti (1999) provides a robust treatment of group theory from the perspective of metaheuristics. Section 2.2 provides a short discussion on metaheuristics, focusing on tabu search (TS). Section 2.3 provides a literature review on the use of group theory in operations research (OR). Section 2.4 reviews recent work on group theoretic metaheuristic search methods.

2.1 An Introduction to Group Theory

2.1.1 Groups

Given a set of elements G and a binary "multiplication" operation $\oplus$ then $\langle G, \oplus \rangle$ is a ***group*** if

1. $\forall g, h \in G \Rightarrow g \oplus h \in G$ (Closure)
2. $g \oplus (h \oplus j) = (g \oplus h) \oplus j \quad \forall g, h, j \in G$ (Associativity)
3. $\exists e \in G \;\; s.t. \; g \oplus e = g \quad \forall g \in G$ (Identity)
4. $\exists g^{-1} \in G \;\; s.t. \; g \oplus g^{-1} = e \quad \forall g \in G$ (Inverse)

An ***abelian*** group embodies a commutative binary operation (Fraleigh 1976, Herstein 1975).

2.1.2 Subgroups

Let G be a group with $H \subseteq G$. If H is also a group under the operation $\oplus$ of G, H is a ***subgroup*** of G denoted $H \leq G$. H must be closed under $\oplus$, contain the identity element e, and contain $g^{-1} \in H \quad \forall g \in H$.

2.1.3 Cosets

If $H \leq G$ and $g \in G$, $gH = \{ g \oplus h,\ h \in H \}$ is a ***left coset*** of H in G and Hg is a ***right coset*** of H in G. The left (right) cosets are disjoint and partition G into equal sized sets. The number of left (right) cosets is $|G|/|H|$ (Colletti 1999). If G is abelian, $gH = Hg \ \forall g \in G$.

2.1.4 Cyclic Groups

For $n > 0$, define g^n as the n-fold product of g (i.e., g multiplied by itself n times). For $n < 0$, g^n is the n-fold product of g^{-1}. Finally, define $g^0 = e$. The ***order*** of g is the minimum positive value of i such that $g^i = g^0 = e$.

Group closure implies if $g \in H$ then $g^i \in H \ \forall i \in \mathbb{Z}$. Let $g \in G$ then the ***cyclic group***, $H = \{ g^n \mid n \in \mathbb{Z} \} = \langle g \rangle$ is the smallest subgroup of G that contains g (Fraleigh 1976). The group $K = \langle g, h, j \rangle$ is the smallest subgroup that contains g, h, and j.

2.1.5 The External Direct Product of Groups

Let G_1, G_2, …, G_n be a collection of groups. Define the group G to be the Cartesian product of the G_i, i.e., the ***external direct product*** of the groups G_i. The binary operation in G is component-wise multiplication: $(g_1, g_2, ..., g_n)(h_1, h_2, ..., h_n) = (g_1h_1, g_2h_2, ..., g_nh_n)$. The identity of G is $(e_1, e_2, ..., e_n)$ and the inverse of any element is $(g_1, g_2, ..., g_n)^{-1} = (g_1^{-1}, g_2^{-1}, ..., g_n^{-1})$ (Fraleigh 1976, Herstein 1975). If all G_i are abelian then G is abelian.

2.1.6 Factor Groups

For $g, h \in G$, define $g^h = h^{-1}gh$ to be the ***conjugate*** of g by h (Fraleigh 1976). If $\exists\, j \in G$ such that $g^j = h$, g and h are conjugates. For abelian G, $g^h = h^{-1}gh = hh^{-1}g = g$ and $\forall g \in G$ the only conjugate element is g itself. $N \leq G$ is a ***normal subgroup*** of G, denoted $N \lhd G$, if $N^g = N \quad \forall g \in G$. For abelian G, all subgroups are normal.

If $N \lhd G$, the left cosets, $gN \ \forall g \in G$, form the ***factor group*** of G modulo N, denoted G / N, under the set product operation (Fraleigh 1976):

$$A \oplus B = \{ab : a \in A, b \in B\} \ \forall A, B \subseteq G$$

2.1.7 Homomorphism

A function f mapping between two possibly distinct groups, $\langle G_1, \oplus \rangle$ and $\langle G_2, \otimes \rangle$, is a ***homomorphism*** if the mapping preserves the group operation. If $f : G_1 \to G_2$ then f is a homomorphism *iff* $f(g \oplus h) = f(g) \otimes f(h)\ \forall g, h \in G_1$. If in addition the homomorphism is a bijection then it is an ***isomorphism***. Two groups which are isomorphic are essentially identical with the elements relabeled. If $N \lhd G$ then the ***natural homomorphism*** is $f : G \to G/N$ mapping each $g \in G$ into the coset gN (Colletti 1999).

2.1.8 Group Action

Given a group G and a set T, the ***group action*** of G on T, ${}_G T$, is a remapping of T such that $\forall g \in G$ and $\forall s \in T$, we have $s \wedge g = t \in T$ (Colletti 1999). Additionally ${}_G T$ must have the following properties:

1. $t \wedge e = t\ \forall t \in T$ (where e is the identity of G)
2. $(t \wedge g) \wedge h = t \wedge (gh)\ \forall t \in T$ and $g, h \in G$

A group action partitions T into disjoint ***orbits***. For group action ${}_G T$, the orbit of $t \in T$ is $\text{Orbit}(G,t) = \{t \wedge g \mid g \in G\}$. If $s, t \in T$ are in the same orbit then $\exists\, g \in G$ such that $s \wedge g = t$.

2.1.9 Examples

S_n has been used in significant recent work in metaheuristics (Colletti 1999, Crino 2002, Wiley 2001). A permutation may be represented by a 2 by *n* matrix corresponding to a 1-1 and onto mapping of the integers {1,2,…,*n*} where the integer in the top row is replaced in the order by its image in the bottom row. For example, if *n*=6 then

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 1 & 2 & 4 & 6 & 5 \end{pmatrix} \qquad q = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 1 & 5 & 3 & 2 & 4 \end{pmatrix}$$

p and *q* are permutations. The set of all permutations of degree *n* along with the binary operation of function composition, $p \oplus q = pq = q(p(x))$, form S_n.

Permutations are often written in cycle notation for example *p* = (132)(4)(56) = (132)(56). Single character cycles or 1-cycles map a letter onto itself and are typically not shown. Cycles with 2 characters (2-cycles) are *transpositions*. The letters moved by *p* are denoted move(p). In chapters 3 and 4, we will use a group action based on a direct product of S_n.

Another important group in this research, $G(D)$, is based on fractions under addition modulo 1. The group elements are *k/D* for some $k, D \in \mathbb{Z},\ 0 \le k < D$. $G(D)$ is closed under addition modulo 1, has identity 0/*D*, and each element *k*/*D* has inverse (*D*-*k*) / *D*. $G(D)$ is abelian, cyclic with generator 1/*D* and has size *D*. Since we can relabel the elements by multiplying each by *D*, the group is isomorphic to the set of integers under addition modulo *D*, denoted *Z*(*D*). The group *Z*(*D*) consists of

the integers {0, 1… *D*-1}. In chapters 5 and 6, we will use a subgroup of an external direct product of *m* of these groups.

2.2 HeuristicMethods

Many discrete optimization problems are NP-hard (Wolsey 1998). Such problems are *difficult*, i.e., polynomial time algorithms to solve these problems do not exist. For such problems, the number of solutions grows exponentially with problem size and all solutions must be explicitly or implicitly enumerated to guarantee optimality. Consequently, heuristic methods are often used to address these types of problems. While no guarantee of solution quality is achieved, empirical results have shown these methods often return acceptable solutions in a very short amount of time.

2.2.1 Local Search Methods

Heuristic methods such as steepest descent, simulated annealing and tabu search are *local search methods* (Glover and Laguna 1997, Reeves 1995). In a local search method, a ***starting solution*** is chosen. A ***move*** modifies the current incumbent solution and all solutions that can be reached in one move comprise the incumbent's ***neighborhood***. The new solution is chosen from the neighborhood based on a merit function and the details of the algorithm. The new solution becomes the incumbent and the procedure repeats until a termination criterion is satisfied.

2.2.2 Tabu Search

In tabu search (TS) (Glover and Laguna 1997), we prohibit recently visited solutions from being revisited for ***tabu tenure*** iterations. At each iteration the best ***non-tabu*** neighboring solution is selected. The tabu memory structure allows escape from local optima to continue the search. TS has been shown to be quite effective in solving complex optimization problems (Crino 2002, Glover and Laguna 1997, Wiley 2001).

Reactive Tabu Search (RTS) was developed by Battiti and Tecchiolli (1994). In early TS implementations, the length of the tabu tenure was fixed or randomly chosen. In RTS, the current tabu tenure is based on the occurrence of repeated visits to the same solutions. When the algorithm detects sufficient repetitions, the tabu tenure is increased to encourage the algorithm to ***diversify*** into a region of the solution space not yet explored.

The tabu memory structure in RTS also helps detect the occurrence of ***cycling*** – revisiting the same solutions repeatedly. When the algorithm detects cycling, an escape procedure is implemented to attempt to break the cycle and hopefully diversify into a region of the solution space not yet explored.

2.2.3 Landscape Theory

An often ignored but very important aspect of local search methods is the problem *landscape*. The landscape structure is determined by the objective function, the solution space and the neighborhood definition (Barnes et al. 2003). The

neighborhood definition determines whether the solution landscape will be favorable for local search.

Weinberger (1990) addresses the idea of correlated and uncorrelated landscapes. He classifies landscapes with approximate decaying exponential autocorrelation spectra to be AR(1) (first order autoregressive) landscapes. Considering only regular and symmetric neighborhoods, Grover (1992) derives a difference equation for the neighborhoods similar to a well known differential equation used in mathematical physics. For landscapes that satisfy Grover's difference equation, all local minima are less than the average solution value over the landscape and all local maxima are greater than the average solution value.

Using group theory, Colletti (1999) proves that symmetric multiple traveling salesmen problems (m-STSPs) with move neighborhoods based on an arbitrary collection of k-city exchange moves will satisfy Grover's difference equation for any value of k.

Again considering only regular and symmetric neighborhoods Stadler (1996) develops his Laplacian and forms the matrix version Grover's equation. He shows that if the normalized objective vector is an eigenvector of his Laplacian matrix then the landscape will satisfy Grover's equation. He calls the associated landscapes ***elementary***. He also claims that for regular symmetric transition matrices, the associated landscape is elementary if and only if it is an AR(1) landscape.

Barnes et al. (2003) extend the theory of elementary landscapes by using a more general Laplacian matrix that also embraces *arbitrary* transition matrices. They

also define two types of elementary landscapes: smooth and rugged. Dimova et al. (2005) extend Stadler's results on AR(1) processes to arbitrary transition matrices. For any transition matrix, the landscape is elementary if and only if the random walk on the landscape follows an AR(1) process.

2.3 Group Theory and Operations Research

2.3.1 Integer Programming and Cutting Planes

Gomory (1963, 1965, 1967, 1969) developed a methodology for adding valid inequalities or cutting planes to an integer linear program (ILP) to drive the relaxed linear program (LP) to an integer solution. The inequalities are formed using the fractional components of an integer combination of the constraint rows in the LP optimal tableau. Each inequality renders the current LP optimum infeasible. In principle, repeatedly adding cuts and resolving will eventually yield the optimal solution to the original ILP.

Gomory (1963) shows:

(1) The fractional components take the form k/D for some $k \in$, $0 \le k < D$ where D is the determinant of the optimal basis to the original LP relaxation.

(2) The ***fractional cuts*** form a group under component-wise addition mod 1.

(3) The group is isomorphic to a factor group M($\boldsymbol{I}$)/M($\boldsymbol{B}$) where M($\boldsymbol{I}$) is the group of all integer vectors of size m and M($\boldsymbol{B}$) is the subgroup of all linear integer combinations of the columns in the optimal LP basis.

Pure cutting plane methodologies fell out of favor in the 1980s due to their lack of convergence in practical implementations. Nevertheless, the combination of cutting planes and branch and bound algorithms (branch & cut) are among the most popular and effective methodologies for solving ILPs and mixed ILPs today.

2.3.2 The Group Minimization Problem

The group minimization problem (GMP) (Gomory 1965, 1967, 1969) is a mathematical formulation for solving ILPs (Johnson 1980) by perturbing the non-basic variables from the associated LP relaxation while ensuring the integrality of the basic variables. Since any such perturbation degrades solution quality, the goal is to minimize such changes. When the GMP yields non-negative basic variables, it solves the original ILP, i.e., an algorithm that solves the GMP also *solves the general* ILP.

The GMP can be expressed using either one of two abelian groups, $H(\boldsymbol{\alpha})$ or M($\boldsymbol{I}$)/M($\boldsymbol{B}$). As alluded to in Section 2.1.9, $H(\boldsymbol{\alpha})$ is a subgroup of an external direct product of m $G(D)$ groups where m is the number of rows in the basis. The group operation is addition modulo 1 and the group is generated by the fractional components of the non-basic columns in the optimal LP tableau. The size of the full group containing $H(\boldsymbol{\alpha})$ is D^m.

M($\boldsymbol{I}$)/M($\boldsymbol{B}$) is the factor group described above. Each non-basic column is mapped to an element of the factor group using the natural homomorphism. The elements then generate the full factor group which also is of size $\leq D$. M($\boldsymbol{I}$)/M($\boldsymbol{B}$) is isomorphic to $H(\boldsymbol{\alpha})$.

Gomory (1969) shows how the GMP can be used to transform the problem into an integer program with one constraint. The problem's decision variables are the integer multipliers for each group element. Early GMP solution approaches (Glover 1968, Shapiro 1968a, 1968b) attempted to enumerate the group using dynamic programming or network flow optimization. Since these approaches could manage only trivially sized GMPs, the GMP was largely forgotten and subsequently ignored by the operations research community.

2.3.3 Corner Polyhedra

Gomory (1967, 1969) described the geometry associated with his cutting planes and GMP. Let P be the feasible region for the LP relaxation of an ILP. Given any vertex of P, we relax all constraints not passing through the vertex to form a cone, K. The convex hull of the integer points in K is the ***corner polyhedra*** (*CP*) for that vertex.

Gomory shows if the vertex is LP optimal then the optimal solution to the ILP is a vertex on the corresponding *CP*. Gomory's fractional cuts are the faces of *CP*. *CP* can be expressed in terms of the original decision variables or in terms of the non-

basic variables. The feasible region for the GMP corresponds to the *CP* in terms of the non-basic variables.

Corner polyhedra may also be expressed in terms of the full group generated by the elements associated with the non-basic columns. This increases the dimension of *CP* to *D*. This is the ***master CP***. Any facet of the *CP* is a facet of the master *CP* and any facet of the master *CP* is a face of the *CP* but not necessarily a facet. Many problems with different *CP* will share a common master *CP*. Gomory (1969) generated facets of the master *CP* in hopes of finding facets to all corresponding *CP*.

2.3.4 Recent Work

Gomory's work with corner polyhedra and master corner polyhedra to generate cutting planes has mostly lain dormant for the last two decades. Recently, Gomory and others have begun to resurrect the research (Gomory and Johnson 2003a, 2003b, Gomory et al. 2003, Aráoz et al. 2003). The focus of that new research is still on cutting planes and the faces of the *CP*. The research documented in this dissertation focuses on the interior of the *CP*.

2.4 Group Theory and Metaheuristics

Although not commonly acknowledged, many aspects of metaheuristics can be defined in terms of group theory. Colletti (1999) gives a comprehensive treatment of group theory in the context of metaheuristics. Using the TSP and TS, he classifies and defines current move definitions in terms of group theory and develops composite

move strategies that would be difficult to generate using other methods. He provides efficient methods for escaping from chaotic attractors during the search and for generating search neighborhoods.

This Group Theoretic Tabu Search (GTTS) approach has been very successfully applied to several complex problems. Wiley (2001) implements Colletti's ideas to solve the Aerial Fleet Refueling Problem. A solution representation is developed using S_n and dynamic move neighborhoods are developed based on conjugation and function composition to explore the solution space. Crino (2002) extended these ideas in solving the military theater distribution problem. He used group actions to partition the solution space into orbits and orbital planes. Our work in chapters 3 and 4 is based on Crino's approach.

The next chapter considers the general partitioning problem and presents a group theoretic strategy for partitioning the solution space to enhance the exploration of that space.

Chapter 3 - Partitioning the Solution Space

Since the solution space for a typical NP-hard problem is immense, appropriate *partitioning* of the solution space can improve the search by aiding in diversification, intensification, and cycle prevention. Group Theory provides many methods for such partitions.

3.1 Partitioning Into Orbits (Ordering Problems)

3.1.1 Theory

Colletti (1999) represents solutions to the single and multiple agent TSPs as elements of S_n and proposes the use of group actions and orbits to partition the solution space. For example, a solution to a 10-city 2-TSP may be (4 6 2 9)(3 1 10 8 5 7) with agent 1 visiting cities 4, 6, 2, and 9 in order and agent 2 visiting cities 3, 1, 10, 8, 5, and 7. Each agent's tour is a cycle and the number and size of the cycles in the solution is the solution's *cycle structure*.

Two elements with the same cycle structure are *conjugates* and are in the same *conjugacy class*. A group action on S_n based on conjugation with $H \leq S_n$ does not change cycle structure. A conjugacy class with b a-cycles and d c-cycles is denoted by $a^b c^d$.

Example 3.1

Let S_{10} be our solution space and let $H = \langle (1\ 2), (1\ 3) \rangle$ ($H \leq S_{10}$). Let the current solution be x = (4 6 2 9)(3 1 10 8 5 7) $\in 4^1 6^1$. The orbit neighborhood for H given x, x_H, is all solutions reachable by rearranging 1, 2, and 3 in x, i.e, all solutions constructed by conjugating x with any element of H:

x^(1 2) = (4 6 1 9)(3 2 10 8 5 7)

x^(1 3) = (4 6 2 9)(1 3 10 8 5 7)

x^(2 3) = (4 6 3 9)(2 1 10 8 5 7)

x^(1 2 3) = (4 6 1 9)(2 3 10 8 5 7)

x^(1 3 2) = (4 6 3 9)(1 2 10 8 5 7)

x^e = (4 6 2 9)(3 1 10 8 5 7)

To change to another orbit with the same cycle structure we can use conjugation with a permutation not in the subgroup H. We must use some other operation, like function composition, to move to an orbit with a different cycle structure. A *transversal* of the orbits is a list containing one element from each possible orbit. Changing the current solution to an element in a different orbit moves the search to that orbit and conjugation with the subgroup H allows the search to explore the orbit. Unfortunately, creating transversals can be quite costly as the number of orbits increases.

Performing the group action on S_n using all of S_n partitions the solution space into orbits each representing a different cycle structure or conjugacy class (i.e. 10^1, $1^1 9^1$, $2^1 8^1$, 2^5, $3^2 2^2$, etc.) . For example, orbit $4^1 6^1$ would contain all solutions

assigning 4 cities to agent 1 and 6 cities to agent 2. Alternatively, we can perform the group action using subgroups of S_n.

Colletti (1999) also discusses the pros and cons of using different sized subgroups of S_n for the group action. A small subgroup yields smaller orbits (each possibly small enough to search exhaustively) but a greater number of such orbits. Using too many orbits demands an effective method to search for an orbit to explore. A larger subgroup yields fewer orbits, likely so large that exploring any orbit is difficult.

3.1.2 Application

Crino (2002, 2004) implements Colletti's ideas to solve the theater distribution routing and scheduling problem (TDVRSP). He partitions the solution space into orbits and further partitions the orbits into sub-orbits. This is accomplished by using a group action on each orbit with a subgroup of the group used to create the orbit. Crino uses a reactive tabu search procedure to search for orbits which are small enough to be exhaustively enumerated. Once an orbit has been explored the orbit is made tabu.

The first partition is based solely on cycle structure or conjugacy class. The second partition is based on *cycle order* (i.e. $2^1 8^1$ vs $8^1 2^1$). Group theory does not distinguish elements of S_n by cycle order $a^b c^d$ is equivalent to $c^d a^b$; however, this distinction is required as the TDVRSP vehicles assigned to each cycle are non-homogeneous.

If the search discovered a good solution with 4 cities assigned to the first vehicle and 6 assigned to the second, it is likely that there are other good solutions in the same orbit. Similarly if the search has trouble finding a feasible solution in orbit $9^1 1^1$, we may begin to believe that the orbit does not contain any feasible solutions and we can abandon it. While we may not know which orbits are good or bad a priori, we may be able to narrow down the good orbits shortly into the search and focus our efforts there.

Crino further partitions the above orbits by assigning customers into sets. The number and size of the sets creates the first partition, an *orbital plane*. The orbital planes are then partitioned into orbits based on which customers are assigned to each set. All of these partitions are created by a group action with some subgroup of S_n. Exchanging 2 or more customers between the sets will move to a new orbit on the orbital plane. Modifying the size or number of sets will move to a new orbital plane.

3.1.3 Observations

Tabu search and other local search heuristics have been shown to be quite effective at finding good solutions quickly without partitioning (Glover and Laguna 1997, Combs 2004). When we partition the solution space, we restrict the movement of the search algorithm. If this is a good restriction, limiting the search to the "good" areas of the solution space, the partitioning can be quite effective. However, arbitrary partitioning or too many partitions may trap the search in bad regions of the solution space.

The first two levels of partitioning used by Crino, cycle structure and cycle order, have some relevance to the problem being solved. It is reasonable to assume that this partitioning scheme would create partitions containing solutions of similar value. The third and fourth levels of partitioning created by grouping the customers into arbitrary sets have no logical link to the problems being solved. One good solution in an orbital plane need not indicate a higher or lower probability of other good solutions in that plane. Superior results could probably be achieved if only the first two levels of partitioning were applied. In general, partitioning the solution space into "good" and "bad" partitions will allow the search to focus on the good partitions.

3.2 Partitioning Into Orbits (Partitioning Problems)

For strict partitioning problems, using S_n for the solution representation is inappropriate. A vector of integers suffices. For binary programs, the solution representation is a binary vector of size n and the solution space, $\boldsymbol{X}$, is the set of all binary vectors of size n.

We can create a group action on $\boldsymbol{X}$ using S_n or a subgroup of S_n and partition the solution space into orbits. Given a permutation $p \in S_n$ and a solution vector x, we define the group action of p on x as moving the value at vector element i to vector element j if j follows i in p. For example, if x = (1 0 1 2 5 0 1 3 1 0) and p = (2 3 4) then $x\text{^}p$ = (1 2 0 1 5 0 1 3 1 0).

This operation is a valid group action because the result of the action is in $\boldsymbol{X}$ and the properties from 2.1.8 hold. Using all of S_n for the group action on $\boldsymbol{X}$ creates only a single orbit. In the next section, we show how to use a subgroup of S_n by clustering the elements of the solution and restricting movement of the values to within the clusters.

3.2.1 Variable Clustering

Grouping the problem variables into clusters and using an external direct product of S_n subgroups creates orbits and partitions the solution space. Let there be n_k variables in cluster k and let r be the number of clusters. The group acting upon the set of n vectors is $S_{n1} \times ... \times S_{nr}$ where $\sum_{j=1}^{r} n_r = n$. For example, if n = 10, n_1 = 3, n_2 = 3, n_3 = 4 and r = 3 a solution may be $\boldsymbol{x} = \left[101\middle|250\middle|1310\right]$. The group acting upon x is $S_{n3} \times S_{n3} \times S_{n4}$ and the orbit containing x contains all solutions where the first cluster contains two 1s and a 0, the second contains a 2, 5, and 0, and the last contains two 1s, a 3 and a 0. The orbit of x is

$\left[101\middle|250\middle|1310\right]$ $\left[110\middle|250\middle|1310\right]$ $\left[011\middle|250\middle|1310\right]$ $\left[101\middle|520\middle|1310\right]$ $\left[110\middle|520\middle|1310\right]$

$\left[101\middle|502\middle|1310\right]$ $\left[110\middle|502\middle|1310\right]$ $\left[011\middle|502\middle|1310\right]$ $\left[101\middle|520\middle|1310\right]$ $\left[011\middle|205\middle|1310\right]$

$\left[101\middle|025\middle|1310\right]$ $\left[110\middle|025\middle|1310\right]$ $\left[011\middle|025\middle|1310\right]$ $\left[101\middle|205\middle|1310\right]$ $\left[110\middle|205\middle|1310\right]$

$\left[110\middle|052\middle|1130\right]$ $\left[110\middle|052\middle|3110\right]$ $\left[011\middle|052\middle|3110\right]$ $\left[101\middle|052\middle|3110\right]$ ...

If we define *cardinality* as the number of times a value appears in the cluster, an orbit can be characterized by the cardinality of the components of each cluster. If a problem is of order *d* (i.e. variable values are 0 thru *d*-1) and has *r* clusters, we can capture the orbit characterization in a *d-1* by *r* matrix, O^p, where element O^p_{ik} is the number times non-zero value *i* appears in cluster *k* in orbit *p*. For example, assume the orbit above is from an order 10 problem, the orbit matrix is

$$\boldsymbol{O}^p = \begin{bmatrix} 2 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

3.2.2 Which Variables Do We Cluster?

Clustering the variables is easy. Since the goal is to group solutions of similar quality in the same orbits, determining how many clusters to create and which variables to assign to each cluster can be more *challenging*.

Often, a logical clustering of the variables is obvious from the context of the problem. For knapsack problems, it seems reasonable to assume that orbits with more non-zero values in clusters with high benefit/cost ratios would be superior to other orbits. If our problem contains special ordered sets (SOSs) then they provide a very natural clustering. For example, if we have 3 SOSs of type 1 (only 1 member of

the set can be non-zero), we create 3 clusters, 1 for each set. Any orbit containing more than 1 non-zero element in each cluster is an infeasible orbit and can be discarded.

Borrowing from the ideas of Gomory (1963, 1965, 1967, 1969), we can solve the LP relaxation of our problem and cluster the variables based on reduced costs. We can place all of the basic variables in the same cluster and cluster the non-basic variables separately based on their reduced costs.

We cannot use reduced costs to explicitly determine which non-basic variables will change values in the near optimal solutions (see Section 3.3.3). However, we can use reduced costs as a heuristic indicator of such changes. To reduce the number of orbits, we place the variables in the same cluster if their reduced costs are within a certain range of each other.

The number of clusters must also be decided and may also be determined by the problem context. If it is not clear how many clusters to create, it is preferable to have too few as opposed to too many. Creating too many clusters leads to too many orbits, and too many orbits restricts the movement of the search making it difficult to find the good regions of the solution space. If we assign each variable to its own cluster, exhaustive enumeration is implied. Alternatively, if we assign all variables to the same cluster, general tabu search is implied. Clearly the latter is preferable.

3.3 Orbit Exploration

3.3.1 Intra-Orbit Move Neighborhoods

If the orbits are small enough we may be able to enumerate them; however, this is not likely to be the case. So we must create a move neighborhood that is of manageable size, restricts the search to the current orbit, and provides connectivity to all solutions within the orbit. Let $G = S_{n1} \times ... \times S_{nr}$ be the group used to create the orbits, we can use any set of elements $H \subseteq G$ that generates G as a move neighborhood possessing these desired properties. H is a subset but not a subgroup of G. A subgroup by definition is closed and therefore no proper subgroup can generate the full group, therefore H cannot be a subgroup of G.

Every element of H is in G and since G is closed all combination of elements of H are also in G. By property 2 of a group action, we know $(x \wedge g) \wedge h = x \wedge (gh)$ $\forall x \in \boldsymbol{X}$ and $g, h \in H$, so executing multiple moves in succession is equivalent to executing one move using the product of the move elements. Therefore the neighborhood based on H restricts the search to the current orbit. Given any solution in the orbit, any other solution in the orbit can be generated using an element of G. Since the elements of H generate G, executing multiple moves will eventually generate all of G and the entire orbit. The size of the neighborhood is simply the size of H.

Every permutation is a product of transpositions and so the set of all transpositions in S_{nk} generates S_{nk}. For G the number of transpositions is $\sum_{k=1}^{r}\binom{n_k}{2}$. This set is essentially a *within-cluster* swap neighborhood (including duplicate solutions). The size of a full neighborhood without regard to the clusters would be a much larger $\binom{n}{2}$. So in addition to partitioning the space the orbit restriction acts as a candidate list restricting neighborhood size.

There are even smaller neighborhoods available. The subset of transpositions containing all transpositions in S_{nk} with the same first element will also generate S_{nk} (i.e. $\{(1\ \ 2),(1\ \ 3),...,(1\ \ n_k)\}$) as will the following set of transpositions $\{(1\ \ 2),(2\ \ 3),...,(n_k-1\ \ n_k)\}$. For G the size of each of these neighborhoods is $\sum_{k=1}^{r}(n_k-1)$.

3.3.2 Inter-Orbit Move Neighborhoods

Any action that changes the orbit's characterization matrix causes the search to leave the current orbit and enter a new one. We can increment or decrement the cardinality of one or more of the non-zero values in one or more clusters. What we cannot do is change the number of clusters or what variables are assigned to each cluster. Doing so would change the partitioning structure as opposed to which partition we are currently exploring.

3.3.3 Bounding the Solutions in an Orbit

We can use the objective function coefficients and the orbit's characteristic matrix to calculate an upper and lower bound on the solutions (not necessarily feasible) in the orbit. Using these bounds, it may be possible to determine if an orbit contains any feasible solutions without actually exploring the orbit. We may also be able to avoid searching an orbit if its solution bounds are dominated by the best solution found so far.

Let c_{jk} be the cost associated with variable j in cluster k. Let nz_k be the number of non-zero elements in cluster k (i.e. the sum of column k in $\boldsymbol{O}^p$). We can calculate the upper bound objective value for each cluster k by performing the following:

1. Sort (within cluster k) the c_{jk} in ascending order
2. Set $j = n_k - nz_k + 1$, $i = 1$, and $z_{UB}^{\,k} = 0$
3. While $O^p_{ik} \neq 0$
   - 3a. $z_{UB}^{\,k} = z_{UB}^{\,k} + i * c_{jk}$
   - 3b. $j = j + 1$
   - 3c. $O^p_{ik} = O^p_{ik} - 1$
4. $i = i + 1$
5. If $i < d$, go to 3.

The upper bound for the orbit is $\sum_{k=1}^{r} z_{UB}^{\,k}$. We can calculate the lower bound of an orbit in the same fashion by traversing the columns of $\boldsymbol{O}^p$ from bottom to top.

Example 3.2

Assume we have the following orbit matrix

$$\boldsymbol{O}^p = \begin{bmatrix} 2 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

with $n = 10$, $n_1 = 3$, $n_2 = 3$, $n_3 = 4$, $d = 10$ and $r = 3$. Let our cost (sorted by cluster then in ascending order) be

$$\begin{bmatrix} 1 & 2 & 4 & 1 & 3 & 5 & 2 & 2 & 4 & 6 \end{bmatrix}$$

The upper bound for cluster 1 is

1. $z_{UB}^{1} = 1*2 = 2$
2. $z_{UB}^{1} = z_{UB}^{1} + 1*4 = 6$

We also have $z_{UB}^{2} = 31$ and $z_{UB}^{3} = 24$ so $z_{UB} = 61$. No solution in the orbit feasible or infeasible will have a solution greater than 61. We also have $z_{LB} = 3 + 11 + 12 = 26$.

We may be able to get tighter bounds by representing the orbit characteristics as additional constraints and solving the LP relaxation for that orbit. Let x_{jk} be the jth variable in cluster k. The constraint or cut associated with cluster k is

$$\sum_{j=1}^{n_k} x_{jk} = \sum_{i=1}^{d-1} O^p{}_{ik}$$

Adding the above cut for each cluster and solving the LP relaxation gives us an upper or lower bound depending on whether we are maximizing or minimizing. If the LP relaxation is infeasible the orbit is infeasible.

If we are maximizing and have an upper bound on our solution from some relaxation method and the lower bound of the orbit is greater than that upper bound, the entire orbit is infeasible and can be discarded. If we have a current best solution greater than the upper bound of the orbit then the orbit is dominated and can be discarded. Similar arguments can be made for minimization.

If we have clustered the variables based on reduced costs we may be able to put a bound on an orbit using the reduced costs. By assigning the non-zero values in a cluster to the smallest reduced costs of the variables in the cluster and adding across all clusters, we can calculate the minimum weighted distance from the LP optimal solution. We subtract this value from the value of the LP optimal solution and we have an upper (max) or lower (min) bound on solutions in the orbit.

3.4 Orbit Size

Not all orbits will be of the same size. The size of the orbit is based on the number of clusters and the number of combinations of the non-zero elements in the clusters. The number of iterations the search spends in an orbit should be a function of orbit size.

Again let d be the order of the vector elements, r the number of clusters and n_k the number of variables assigned to cluster k. For orbit p cluster k, the number of

ways to assign the first non-zero value is $\binom{n_k}{O^p{}_{1k}}$. After assigning the first non-zero value the number of ways to assign the second is $\binom{n_k - O^p{}_{1k}}{O^p{}_{2k}}$ and so on. So the number of combinations for orbit *p* cluster *k* is

$$\prod_{i=1}^{d} \binom{n_k - \sum_{m=1}^{i-1} O^p{}_{mk}}{O^p{}_{ik}}$$

The number of solutions in orbit k is the product of the number of combinations for each cluster in the orbit. So the size of the orbit is

$$\prod_{k=1}^{r} \left(\prod_{i=1}^{d} \binom{n_k - \sum_{m=1}^{i-1} O^p{}_{mk}}{O^p{}_{ik}} \right)$$

Example 3.3

For the following orbit

$$\boldsymbol{O}^p = \begin{matrix} \lceil 2 & 0 & \overline{2} \rceil \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \lfloor \underline{0} & 0 & \underline{0} \rfloor \end{matrix}$$

the size is $\binom{3}{2} \bullet \binom{3}{1}\binom{2}{1} \bullet \binom{4}{2}\binom{2}{1} = 216$.

3.5 Orbit Landscapes

As stated in Section 2.2.3, if a landscape is elementary all local optima are at least better than the average solution. The orbits partitioning the solution space are distinct. While they may share the objective function and neighborhood definition, each orbit has its own set of solutions. A neighborhood that creates an elementary landscape in each orbit implies all of the orbit's local optima are better than the *orbit's* average solution. This can greatly enhance the search if the solution space is partitioned into "good" and "bad" orbits.

3.6 Escaping Attractors

Orbits also provide a deterministic method of escaping chaotic attractors (Colletti 1999). Battiti and Tecchiolli (1994) describe chaotic attractors as landscape characteristics that keep the search confined to a limited region of the solution space. Reactive tabu search is designed to detect and escape from such regions. Such escape procedures can cause a significant change in the current solution. Search methods like those described above can escape the attractor simply by moving to a different orbit.

3.7 Parallel Exploration

Since the orbits are disjoint, this type of partitioning lends itself well to parallel search. A centralized control algorithm could generate orbit characteristic matrices and pass them to available parallel processors. After the search of an orbit is

completed, based on stopping criteria, the processor could return the results to the control algorithm and be assigned a new orbit.

3.8 Conclusions

Partitioning the solution space into orbits allows the algorithm to intensify the search in areas of the solution space believed to contain good solutions. The orbits keep the search contained in these areas and the clusters work as an enhanced candidate list, reducing the total number of moves in the neighborhood while still retaining the "good" moves. In the next chapter, we present a reactive tabu search implementation of these concepts for the unicost set covering problem.

Chapter 4 - A Group Theoretic Tabu Search Algorithm for Unicost Set Covering Problems (GTTS-USCP)

In this chapter we develop a group theoretic tabu search (GTTS) algorithm for solving the unicost set covering problem (SCP), the GTTS-USCP. We solve a linear programming (LP) relaxation of the problem and use the LP optimum to construct a quality solution profile. As described in the previous chapter, we use group theory to partition the solution space into orbits based on this profile. We tested our algorithm on 65 benchmark problems and compared the results against the previous best known and solutions obtained by CPLEX 9.0. GTTS-USCP discovered 46 new best known solutions. GTTS-USCP converged *significantly* faster than CPLEX for all problem sets.

4.1 Problem Definition and Historical Background

The set covering problem (SCP) is a well-known combinatorial optimization problem. Given a 0-1 incidence matrix $\boldsymbol{A}$ with m rows and n columns, the problem is to select the minimum weight subset of columns while ensuring every row is covered. Formally, the problem is:

$$\begin{aligned} &\text{Minimize} && z = \sum_{j=1}^{n} w_j x_j && && \\ &\text{Subject to} && \sum_{j=1}^{n} a_{ij} x_j \geq 1 && i = 1,\ldots,m && \text{(P1)} \\ & && x_j \in \{0,1\} && j = 1,\ldots,n && \end{aligned}$$

If a_{ij} = 1, column *j covers* row *i* and w_j is the weight or cost of column *j*, $\boldsymbol{a}_j$. If $\boldsymbol{a}_j$ is selected to be in the subset, x_j is set to 1 and w_j is added to the cost of the solution. When w_j = 1 for all *j*, the problem is the *unicost* SCP (Grossman and Wool 1997). The cardinality of any P1 solution, $\boldsymbol{x}$, is the number of x_j =1, which, in the case of the unicost SCP, is the value of the objective function, *z*. The linear programming (LP) relaxation of P1 where the x_j can be non-integer is denoted by P1 which has optimal solution, $\boldsymbol{x}^*_{\mathrm{LP}}$, and optimal objective function, z^*_{LP}.

The SCP has many practical applications including crew scheduling (Balas and Carrera 1996, Ceria et al. 1998, Combs 2002, Combs and Moore 2004), emergency facility location (Daskin and Stern 1981, Toregas et al. 1971) and political redistricting (Garfinkel 1970). The SCP is known to be NP-Hard and both exact (Balas and Carrera 1996, Balas and Ho 1980, Beasley 1987, Daskin and Stern 1981, Garfinkel 1970, Toregas et al. 1971) and heuristic (Beasley 1987, Beasley and Chu 1996, Ceria et al. 1998, Chvatal 1979, Grossman and Wool 1997) approaches have been proposed for it. Several of these have made extensive use of Lagrangian relaxation (Balas and Carrera 1996, Balas and Ho 1980, Beasley 1987, Beasley and Chu 1996, Ceria et al. 1998) and column dominance; neither of which are as effective for unicost SCPs.

Grossmann and Wool (1997) explore the performance of nine different heuristic algorithms on the unicost SCP. Most algorithms tested were LP rounding-based and construction-based approaches. A neural network algorithm was also

included in the study. A randomized version of the greedy construction algorithm (Chvatal 1997, Johnson 1974) produced the best results.

4.2 Methodology

4.2.1 Overview

We propose to solve the unicost SCP with a group theoretic tabu search (GTTS-USCP) algorithm. Local search heuristics provide a distinct advantage in solving the unicost SCP as they allow the search to be based on the cardinality of the solution. As described in chapter 3, we use group theory to partition the solution space into orbits based on a *solution profile* created from the optimal solution to the LP relaxation. The LP relaxation of an IP is used extensively in exact methods, but with the exception of the LP bound, is not commonly used in metaheuristics. For the SCP, the P1 solution provides valuable information for a direct search approach. The choice of the non-basic variables, $\boldsymbol{x}_N$, the basic variables, $\boldsymbol{x}_B$, and the reduced costs of the $\boldsymbol{x}_N$, $\bar{\boldsymbol{c}}_N$, can provide valuable insight into the characteristics of high-quality IP solutions.

The methodology is presented in detail in the sections that follow. However, for the purposes of clarity and ease of understanding, we provide a global overview here. First, CPLEX solves P1. The associated optimal basic variables, the x_{Bi}, and the $\bar{\boldsymbol{c}}_N$ are used to assign the x_j to *clusters*. A TS *starting solution,* $\boldsymbol{x}^0$, is generated

by setting $\boldsymbol{x}_B = \mathbf{0}$ and then *selecting* a subset of the x_{Bi} to raise to value 1 so that all rows are covered in P1. A RTS procedure is used to explore the orbits with cardinality between that of $\boldsymbol{x}^0$ and $\boldsymbol{x}^*_{\mathrm{LP}}$. A separate RTS procedure is also used to find *quality* orbits to explore. The algorithm terminates when a GTTS-USCP solution value equals the P1 lower bound, $z_{\mathrm{LB}} = \lceil z^*_{\mathrm{LP}} \rceil$ or when the allotted time has expired.

4.2.2 Clustering Variables Based on the LP Relaxation

The CPLEX solution to P1 details z^*_{LP}, $\boldsymbol{x}_B$, $\bar{\boldsymbol{c}}_N$, and which $\boldsymbol{x}_N$ are at their upper (1) or lower (0) bounds. All x_{Bi} join the same cluster. The upper bound variables, x_{NUj}, and lower bound variables, x_{NLj}, cluster separately based on their reduced costs, $\bar{c}_{NUj}$ and $\bar{c}_{NLj}$. For unicost P1, all $\bar{c}_{NLj} \leq 1$. If $\bar{c}_{NLj} = 1$, $\boldsymbol{a}_j$ does not cover any of rows associated with the binding constraints at optimality. A $\bar{c}_{NUj} = -1$ implies that if $\boldsymbol{a}_j$ is removed from the solution, we must *select* at least two new columns to render the LP solution feasible. While it is possible for $\bar{c}_{NUj}$ to be less than -1, it is quite unusual. To avoid creating too many clusters and too many orbits, we place the x_{Nj} in the same cluster if their $\bar{c}_{Nj}$ are equal to the nearest 0.1 digit. As illustrated in Figure 4.1, the clusters are created in the following order: upper bound variables, basic variables, lower bound variables.

Upper bound						Basic	Lower bound					
…	…	…	-0.2	-0.1	-0.0	0.0	0.0	0.1	0.2	…	…	1.0

Figure 4.1 – Variable

We initially set all $x_{NUj} = 1$, all $x_{NLj} = 0$ and all $x_{Bi} = 0$ making the solution integer feasible. Integer feasibility is then *maintained* throughout the algorithm. Since the problem is binary, the elements of the solution are order 2. The orbit characterization matrix is then a 1 by *r* vector, where *r* is the number of clusters. For example, a solution to a 22 variable problem could be

$$\left[1111\middle|1101\middle|10100\middle|001\middle|000000\right]$$

where the solution cardinality is *z* = 10 and the orbit characterization vector is $\boldsymbol{O}^p = \begin{pmatrix} 4 & 3 & 2 & 1 & 0 \end{pmatrix}$.

4.2.3 Partitioning the Solution Space into Orbits

The solution space is partitioned into orbits first by total solution cardinality, *z*, and second by the cardinality of the individual clusters as described in $\boldsymbol{O}^p$. Initially all orbits are *open*. A list of orbits visited is maintained and an orbit is *closed* to further search after it has been searched for MAX_ORBIT_ITER iterations or has been visited MAX_ORBIT_VISITS times. An orbit hash function (Woodruff and Zemel 1993), $\Omega(k)$, is used to facilitate access to orbit information. Orbit hash values are calculated by generating a random number, τ_j, for each cluster *j*. The orbit's hash value is $\Omega(\boldsymbol{O}^p) = \sum_{j=1}^{r} \tau_j O^p{}_j$. Orbits can be further distinguished by the

number of *free ones* and *free zeros* in the orbit. Free ones (zeroes) are the number of ones (zeroes) in clusters that are not all ones (zeroes). For example, the solution below has 6 free ones and 6 free zeroes.

$$\left[1111\middle|1101\middle|10100\middle|001\middle|000000\right]$$

Any orbit with $\sum_{j=1}^{r} O^p{}_j$ less than z_{LB} contains only infeasible solutions and may be immediately closed. Once a feasible solution is found with cardinality z_{UB}, all orbits with $\sum_{j=1}^{r} O^p{}_j \geq z_{\mathrm{UB}}$ are dominated and may be closed. It is probable that orbits with large $O^p{}_j$ in the upper bound clusters and small $O^p{}_j$ in the lower bound clusters will contain good solutions. This partitioning scheme permits the concentration of search effort within these good orbits.

4.2.4 Inter- and Intra-Orbit Neighborhoods

As shown in Section 3.3.1, our group acting on the set of binary vectors size *n* is $G = S_{n1} \times \ldots \times S_{nr}$ where $\sum_{j=1}^{r} n_r = n$. For our intra-orbit neighborhood we need a set $H \subseteq G$ such that $G = \langle H \rangle$. We let *H* equal the set of all transpositions in *G*. This is equivalent to all moves swapping two elements in the same cluster.

Any move that changes $\boldsymbol{O}^p$ is an inter-orbit move. We use 3 different inter-orbit neighborhoods at different points in the GTTS-USCP algorithm. $O^p{}_j$ is increased (*select neighborhood*) or decreased (*unselect neighborhood*) by "toggling" the value of a single cluster *j* variable. Both $O^p{}_j$ and $O^p{}_k$ may be changed by

swapping values of variables in clusters *j* and *k*. When searching inter-orbit neighborhoods we consider only open orbits.

For all neighborhoods, the best move is defined in terms of *deficit*, the number of unsatisfied rows, and *surplus*, the total amount that rows are over-satisfied. A row is over-satisfied when it is covered more than once. A row covered by 3 columns has a surplus of 2. Once a feasible solution is found we decrease *z* by unselecting a column, therefore GTTS-USCP is usually searching an orbit for feasible solutions. The best move yields the smallest deficit. Deficit ties are broken by largest surplus and surplus ties are broken by order of evaluation (lexicographically).

A *complete* swap neighborhood (which ignores cluster membership) would be $O(n^2)$. For intra-orbit swaps, this effort is reduced by considering swap pairs only if they are in the same cluster. Further, if $O^p{}_j = n_j$ or $O^p{}_j = 0$ no swaps are considered in cluster *j*. However, even with this reduction a complete within-cluster or outer-cluster swap neighborhood is too costly for large problems. Both the intra-orbit and inter-orbit swap neighborhoods are based on a *conditional select move*. First, moves that unselect a non-tabu selected column in the current solution are evaluated. The best of such moves is chosen. *Given* that chosen column will be unselected, we next find the best non-tabu unselected column to select.

A complete select neighborhood is $O(n)$. This neighborhood's size may be reduced by incorporating a candidate list heuristic that also helps diversify the search. A column selection is considered only when the current solution is infeasible. We

first find the row that has been unsatisfied for the longest number of iterations, then we only select from the columns that will satisfy that row.

Since ties in surplus are broken lexicographically, the order in which the clusters are evaluated affects the performance of the algorithm. For select neighborhoods, we first examine the upper bound clusters then the basic cluster and finally the lower bound clusters. This favors increasing the cardinality of the upper bound clusters over the rest. Unselect neighborhoods are examined in the opposite order of select neighborhoods favoring decreasing the cardinality of the lower bound clusters over the rest.

4.2.5 Tabu Lists and Tabu Tenure

Two types of tabu structures are used in our algorithm. The first is a broad-gauge structure which tracks the last iteration a column was selected or unselected. A selected or unselected column's status cannot be changed again for tabu tenure iterations. The tabu tenure may increase or decrease when the algorithm detects cycling.

The second type of tabu structure is a fine-gauge structure which tracks each *individual solution*, noting when it was last visited, and how many times it has been visited. For efficiency, another hash function (Woodruff and Zemel 1993), $\varphi(\boldsymbol{x})$, is used. A random number, ρ_j, is generated for each x_j. The solution's hash value is

$\varphi(\boldsymbol{x}) = \sum_{j=1}^{n} \rho_j x_j$. Solutions can be further distinguished by their deficit and surplus.

Each orbit maintains its own tabu structures. In addition, a separate tabu structure is used during the RTS procedure to find quality orbits. The default tabu tenure for select moves is SELECT_TENURE and the default tabu tenure for unselect moves is UNSELECT_TENURE.

The solution tabu structure is used to detect cycling and to control the tabu tenures. If a solution is repeated, the tabu tenures are increased by a multiplicative factor (*1.618). If MIN_NEW_SOLS consecutive new solutions are visited, the tabu tenures are returned to their default values. If MAX_REPEATED_SOLS solutions are repeated MAX_REPEATS times, the search is presumed to be in a chaotic attractor basin (Battiti and Tecchiolli 1994) and an *escape* is achieved by departing the current orbit or by increasing the tabu tenure if we are still searching for an orbit.

4.2.6 Finding a Starting Solution

To obtain a starting solution, all x_{Nj} are fixed at their values in $\boldsymbol{x}^*_{\mathrm{LP}}$ and the x_{Bi} are set to zero. As illustrated in the pseudo code of Figure 4.2, basic columns are then selected until z_{LB} is reached. If that solution is not feasible, intra-orbit swap moves are performed within the basic cluster, until no improving move is available (steepest descent). If the solution is still not feasible, a select move is performed within the basic cluster and the process repeats until feasibility is achieved. Since

selecting all columns would certainly be a feasible, albeit poor, solution we will converge to a feasible solution in a maximum of n - z_{LB} steps. After feasibility is achieved, any identified redundant columns, columns whose *unselection* will not destroy feasibility, are removed.

```
Generate Feasible Starting Solution
    Select columns from the basic cluster until the z_LB is reached
    If feasible
        Terminate with optimal solution
    Else {
        While not feasible {
            Select a column from the basic cluster
            While improving move found
                Execute intra-orbit swap move
        }
        Attempt to unselect redundant columns
    }
    Save the solution and z_UB
```

Figure 4.2 – Starting Solution Algorithm

4.2.7 Primary Search Strategy

A pseudo-code of the *primary search* strategy is presented in Figure 4.3. After the initial solution is obtained, the GTTS-USCP still focuses on the basic variables. A non-basic variable's status is modified only if doing so achieves feasibility. Since the *duality gap*, z_{UB} - z_{LB}, is known, $\lceil (z_{UB} - z_{LB})/2 \rceil$ columns are unselected from the basic cluster. Next, an RTS procedure, based on the intra-orbit swap neighborhood, is applied to the resulting orbit until (1) a feasible solution is found, (2) MAX_ORBIT_ITER iterations have been performed, (3)

MAX_NI_ORBIT_ITER iterations have been performed without improving the best orbit solution, or (4) the time limit is reached. If a feasible solution is obtained, the duality gap is recalculated, $\lceil (z_{UB} - z_{LB})/2 \rceil$ columns are unselected from the basic cluster and the process repeats.

If a feasible solution is not found, the search departs the current orbit and the orbit's best solution is instantiated as the current incumbent solution. Next, the current inter-orbit swap neighborhood is evaluated in pursuit of a move leading to a feasible solution. If that search yields feasibility, the duality gap is recalculated and the process repeats; if not, the current select neighborhood is evaluated. If feasibility is achieved, the duality gap is recalculated and the process repeats. If both neighborhoods fail to find a feasible solution, a basic cluster column is selected and the new orbit is searched. If z_{UB} is reached without achieving feasibility, the search is expanded.

Primary Search

Search Orbit with a RTS procedure

Perform intra-orbit swap moves until a feasible solution is found or termination criteria is met

If feasible {

Attempt to unselect redundant columns

Save the solution and z_{UB}

If $z_{UB} = z_{LB}$

Terminate with optimal solution

Else {

Remove $(z_{UB} - z_{LB})/2$ columns from the *basic* cluster

Goto ***Primary Search***

}

}

Else {

If feasible inter-orbit swap move found {

Execute swap

Goto ***Primary Search***

}

Else If $z + 1 < z_{UB}$ {

If feasible select column move found {

Select the column

Goto ***Primary Search***

}

Else {

Select a column from the *basic* cluster

Goto ***Primary Search***

}

}

Else

Goto ***Expanded Search***

}

Figure 4.3 – Basic Search Algorithm

```
Expanded Search
    Search for a feasible solution with a RTS procedure
        Perform intra-orbit and inter-orbit swap moves until a feasible solution
        is found or termination criteria is met
    If feasible {
        Attempt to unselect redundant columns
        Save the solution and z_UB
        If z_UB = z_LB
            Terminate with optimal solution
        Else {
            Unselect a column
            While feasible solution found {
                Search Orbit with Reactive Tabu Search
                    Perform intra-orbit swap moves until a feasible
                    solution is found or termination criteria is met
                If feasible {
                    Attempt to unselect redundant columns
                    Save the solution and z_UB
                    If z_UB = z_LB
                        Terminate with optimal solution
                    Unselect a column
                }
            }
            Goto Expanded Search
        }
    }
```

Figure 4.4 – Expanded Search Algorithm

4.2.8 Expanding the Search

A pseudo-code of the *expanded search* strategy is presented in Figure 4.4. After exploring the orbits near the optimal LP solution, GTTS-USCP expands the search to other areas of the solution space. A RTS procedure, based on both inter-orbit and intra-orbit swap neighborhoods, is used to find a good region for

exploration. This RTS procedure continues until either a feasible solution is found or the time limit is reached.

If a feasible solution is found, we unselect a single column (from any cluster) and explore the resulting orbit. If a feasible solution is found while exploring the orbit, we unselect a column again and the process repeats. If a feasible solution is not found while exploring the orbit, we begin another RTS procedure at the new *z*. The process ends when the time limit is reached or a solution equal to z_{LB} is found.

4.3 Computational Results

4.3.1 Test Cases

The benchmark problems solved were obtained from Beasley's OR-Library (Beasley 1990). Problem sets 4-6 originally appeared in (Balas and Ho 1980), problem sets A-D appeared in (Beasley 1987) and problem sets NRE-NRH appeared in (Beasley 1990). All problems were randomly generated based on the strategy of (Balas and Ho 1980). All of these problems were generated as weighted SCPs. Grossman and Wool (1997) solved all but NRG and NRH as unicost SCPs. One problem set, E from (Beasley 1987), is unicost but is not solved here due to its trivial size (all algorithms reach an optimal solution in 0 seconds).

4.3.2 Test Procedures

All tests were performed on Dell Precision 530 Workstations running SuSE Linux with two 1.8GHz Pentium Xeon processors and 1GB of RAM. The machines are multi-user platforms. An attempt was made to find machines that were not too busy, but as each problem ran for at least an hour, CPU usage surely fluctuated during processing. Each problem was solved using CPLEX 9.0 and GTTS-USCP. The time limits used were 7200 seconds for CPLEX and 3600 seconds for GTTS-USCP. The GTTS-USCP algorithm was coded in C.

The previously published best known solutions for these problems were published in Grossman and Wool (1997). They performed their tests on an IBM RS6000 model 370 workstation with 128MB of RAM. They also coded their algorithms in C.

4.3.3 CPLEX 9.0

The algorithms tested by Grossman and Wool (1997) are unsophisticated by today's standards. To provide a more modern benchmark, we compare our results to CPLEX version 9.0. CPLEX uses a very sophisticated branch and cut algorithm to solve mixed integer programs (MILPs). The algorithm is further aided by two heuristics. The first attempts to create a feasible solution from the fractional solution at the node. The second attempts to improve the incumbent integer solution through a neighborhood search (ILOG 2003).

CPLEX was also used to solve P1 for the GTTS-USCP algorithm. The dual simplex LP solver was used for the smaller problem sets, 4-6 and A-D. The sifting LP solver was used for the larger problem sets, NRE-NRH. The default settings were used for all other parameters.

4.3.4 Results

Tables 4.1 and 4.2 contain the results of our tests as well as the problem details and previous best known solutions. The best solution found for each problem is highlighted in bold. GTTS-USCP found the best solution on 59 of the 65 problems. It outperformed CPLEX on 47 of 65. It significantly outperformed CPLEX on the larger problem sets NRG and NRH. Curiously, neither CPLEX nor GTTS-USCP were able to do as well as R-Gr (Grossman and Wool 1997) on the problem sets with higher density, NRE and NRF. None of the solutions were proven to be optimal.

It should be noted here that CPLEX was executed using the default settings while GTTS-USCP, as well as the algorithms from Grossman and Wool (1997), were specifically designed to solve the unicost SCP. A researcher well-versed in CPLEX's parameters and settings would likely be able to improve its performance through experimentation. However, the disparity in performance between CPLEX and GTTS-USCP is quite dramatic and we do not believe any such improvements would be enough to close this gap.

problem	number of rows	number of columns	density	previous best known	CPLEX 9 best (seconds)	GTTS-USCP best (seconds)
4.1	200	1000	2%	41	**38** (11)	**38** (938)
4.2	200	1000	2%	38	**37** (42)	**37** (5)
4.3	200	1000	2%	40*	**38** (28)	**38** (1)
4.4	200	1000	2%	41	39 (851)	**38** (272)
4.5	200	1000	2%	40	39 (64)	**38** (23)
4.6	200	1000	2%	40	38 (50)	**37** (3)
4.7	200	1000	2%	41	39 (211)	**38** (413)
4.8	200	1000	2%	40	**38** (131)	**38** (6)
4.9	200	1000	2%	40	**38** (835)	**38** (35)
4.10	200	1000	2%	41	**38** (1772)	**38** (161)
problem set 4 average				40.2	38.2 (399.5)	37.8 (185.7)
5.1	200	2000	2%	**35**	**35** (824)	**35** (5)
5.2	200	2000	2%	**35**	**35** (137)	**35** (6)
5.3	200	2000	2%	36	35 (373)	**34** (39)
5.4	200	2000	2%	36	35 (122)	**34** (1182)
5.5	200	2000	2%	36	35 (2257)	**34** (12)
5.6	200	2000	2%	36	36 (29)	**34** (989)
5.7	200	2000	2%	35*	35 (33)	**34** (75)
5.8	200	2000	2%	37	35 (85)	**34** (74)
5.9	200	2000	2%	36	36 (113)	**35** (6)
5.10	200	2000	2%	36	35 (4739)	**34** (1873)
problem set 5 average				35.8	35.2 (871.2)	34.3 (426.1)
6.1	200	1000	5%	**21**	22 (3)	**21** (5)
6.2	200	1000	5%	**21**	**21** (2)	**21** (6)
6.3	200	1000	5%	**21***	22 (3)	**21** (10)
6.4	200	1000	5%	22	22 (40)	**21** (4)
6.5	200	1000	5%	22	22 (2)	**21** (25)
problem set 6 average				21.4	21.8 (10)	21 (10)

Table 4.1 – Results for problem sets 4-6
* - Solution found by Neural Network algorithm (Grossman and Wool 1997)

problem	number of rows	number of columns	density	previous best known	CPLEX 9 best (seconds)	GTTS-USCP best (seconds)
A.1	300	3000	2%	40	42 (19)	**39** (337)
A.2	300	3000	2%	41	41 (21)	**39** (79)
A.3	300	3000	2%	40	40 (398)	**39** (179)
A.4	300	3000	2%	40	42 (19)	**38** (1715)
A.5	300	3000	2%	40	39 (4598)	**38** (771)
problem set A average				40.2	40.8 (1011)	38.6 (616.2)
B.1	300	3000	5%	23	23 (826)	**22** (719)
B.2	300	3000	5%	**22**	23 (134)	**22** (17)
B.3	300	3000	5%	**22**	23 (12)	**22** (698)
B.4	300	3000	5%	23	23 (77)	**22** (1910)
B.5	300	3000	5%	23	23 (2422)	**22** (46)
problem set B average				22.6	23 (694.2)	22 (678)
C.1	400	4000	2%	45	48 (3251)	**43** (1524)
C.2	400	4000	2%	45	46 (4488)	**44** (197)
C.3	400	4000	2%	45	49 (41)	**43** (1029)
C.4	400	4000	2%	46	47 (3994)	**43** (1325)
C.5	400	4000	2%	45	48 (6006)	**44** (149)
problem set C average				45.2	47.6 (3556)	43.4 (844.8)
D.1	400	4000	5%	26	27 (2823)	**25** (395)
D.2	400	4000	5%	**25**^	26 (32)	**25** (1890)
D.3	400	4000	5%	**25**	26 (445)	**25** (91)
D.4	400	4000	5%	26	26 (178)	**25** (226)
D.5	400	4000	5%	26	26 (3489)	**25** (200)
problem set D average				25.8	26.2 (1393.4)	25 (560.4)
NRE.1	500	5000	10%	**17**	18 (77)	18 (38)
NRE.2	500	5000	10%	**17**	18 (77)	18 (27)
NRE.3	500	5000	10%	**17**	18 (455)	18 (32)
NRE.4	500	5000	10%	**17**	18 (71)	**17** (54)
NRE.5	500	5000	10%	**17**	**17** (586)	18 (176)
problem set NRE average				17	17.8 (255.2)	17.8 (65.4)
NRF.1	500	5000	20%	**10**	11 (90)	11 (29)
NRF.2	500	5000	20%	**11**	**11** (94)	**11** (39)
NRF.3	500	5000	20%	**11**	**11** (132)	**11** (30)
NRF.4	500	5000	20%	**11**	**11** (1083)	**11** (22)
NRF.5	500	5000	20%	11	**10** (482)	11 (23)
problem set NRF average				10.8	10.8 (376.2)	11 (28.6)
NRG.1	1000	10000	2%	-	74 (582)	**63** (1089)
NRG.2	1000	10000	2%	-	76 (175)	**61** (3401)
NRG.3	1000	10000	2%	-	75 (6426)	**62** (901)
NRG.4	1000	10000	2%	-	74 (3723)	**63** (1045)
NRG.5	1000	10000	2%	-	73 (577)	**63** (406)
problem set NRG average				-	74.4 (2296.6)	62.4 (1368.4)
NRH.1	1000	10000	5%	-	40 (756)	**35** (2008)
NRH.2	1000	10000	5%	-	39 (5716)	**36** (297)
NRH.3	1000	10000	5%	-	39 (1527)	**36** (968)
NRH.4	1000	10000	5%	-	40 (748)	**35** (940)
NRH.5	1000	10000	5%	-	37 (5734)	**36** (454)
problem set NRH average				-	39 (2896.2)	35.6 (933.4)

Table 4.2 – Results for problem sets A-D and NRE-NRH
^ - Solution found by Alternating Greedy algorithm (Grossman and Wool 1997)

Since GTTS-USCP and CPLEX were executed on the same platform, we can make an accurate comparison of the convergence properties. For the purposes of this dissertation we define convergence as the time or effort required to reach a similar level of solution quality. Table 4.3 shows the average CPU seconds for each problem set for GTTS-USCP and CPLEX. Since the algorithms achieved different quality solutions, the time to the best common solution is used for the comparison whenever possible. When no common quality solution exists, the next lower solution for GTTS-USCP is used. GTTS-USCP converged significantly faster than CPLEX for all problem sets.

Table 4.4 compares the convergence properties of GTTS-USCP to R-Gr (Grossman and Wool 1997). R-Gr was executed on a slower computer and for 5 of the problems GTTS-USCP did not reach the same quality solution. As expected R-Gr is faster than GTTS-USCP; however, GTTS-USCP significantly outperforms R-Gr in terms of overall solution quality. If R-Gr was executed for a longer period of time or for more iterations, it is unlikely to improve upon the solutions it has already found.

problem set	**CPLEX 9 avg (sec)**	**GTTS-USCP avg (sec)**
4	38.2 (399.5)	38.2 (116.6)
5	35.2 (871.2)	35.2 (4.4)
6	21.8 (10)	21.8 (2.8)
A	40.8 (1011)	40.8 (21.2)
B	23 (694.2)	23 (26.8)
C	47.6 (3556)	47.4 (7.2)
D	26.2 (1393.4)	26 (46.2)
NRE	18 (150.6)	18 (57.8)
NRF	11 (296.2)	11 (27)
NRG	74.4 (2296.6)	68.6 (114.4)
NRH	39 (2896.2)	38.2 (175)

Table 4.3 – CPLEX vs GTTS-USCP solve time

problem set	R-Gr Avg (sec)	GTTS-USCP avg (sec)
4	40.3 (1.6)	39.6 (0.8)
5	35.9 (3)	35.8 (2.6)
6	21.6 (2)	21.6 (3.4)
A	40.2 (6)	40 (7.8)
B	22.6 (8)	22.6 (167.4)
C	45.2 (10)	45.2 (20.8)
D	25.8 (14.2)	25.8 (54.8)
NRE	17 (38)	17.8 (65.4)
NRF	10.8 (71.2)	11 (27)

Table 4.4 – R-Gr vs GTTS-USCP solve time

4.4 Conclusion

The use of variable clustering and group theory allowed our algorithm to intensify the search in the areas of the solution space believed to contain good solutions. The orbits kept the search contained in these areas and the clusters worked as an enhanced candidate list, reducing the total number of moves in the neighborhood while still retaining the "good" moves. These techniques proved very effective for the unicost SCP discovering 46 new best known solutions to the benchmark problems. However, these techniques are very problem dependent. In the next chapter we examine group theoretic local search techniques for the general IP.

Chapter 5 - Local Search Neighborhoods for General IP

The feasible region for a linear program (LP) is a convex polyhedron (*P*) which may be bounded or unbounded. If the LP decision variables are constrained to be integer, the result is an integer linear programming problem (ILP). The convex hull of *P*, conv(*P*), is the smallest polyhedron containing the integer points of *P*. If we can define conv(*P*), we can relax the ILP integrality requirements and solve the corresponding LP. Our solution will be integer and we will have the optimal solution to the original ILP. Unfortunately defining conv(*P*) is impractical for most problems.

Often we are able to find a feasible integer solution within *P*. We can perform a local search by starting from this solution and moving to an adjacent feasible point by increasing or decreasing a variable value and checking the constraint set to ensure we remain in *P*.

The group minimization problem (GMP) developed by Gomory (1965, 1967, 1969) is the mathematical formulation for deriving the optimal ILP solution from the optimal solution to the corresponding LP (Johnson 1980). Under certain conditions (detailed below), the optimal solution to the GMP is the optimal solution to the original ILP. When we solve the GMP, we solve the ILP and vice-versa. Furthermore, the form of the GMP is the same regardless of the context of the original ILP, so an algorithm to solve the GMP can also solve any ILP.

The feasible region for the GMP is also a convex polyhedron, called the corner polyhedron (*CP*). The objective function costs in the GMP are the reduced costs of the non-basic variables from the optimal LP solution. These costs are the

penalties for moving away from the LP optimal point, $\boldsymbol{x}^*_{\text{LP}}$ along each non-basic variable's axis. So $\boldsymbol{x}^*_{\text{LP}}$ acts as an anchor point for our search. By examining the local search in terms of the GMP, we gain insights that will help us solve the ILP.

5.1 The Group Minimization Problem (GMP)

5.1.1 Derivation of the GMP

Given an ILP, we derive the associated GMP by first solving the LP relaxation. Consider the ILP (5.1).

$$\begin{aligned} \text{Max} \quad & \hat{\boldsymbol{c}}\hat{\boldsymbol{x}} \\ \text{s.t.} \quad & \hat{\boldsymbol{A}}\hat{\boldsymbol{x}} \le \boldsymbol{b} \\ & \hat{\boldsymbol{x}} \ge \boldsymbol{0} \text{ and integer} \end{aligned} \tag{5.1}$$

where $\hat{\boldsymbol{c}}$ is a size n row vector, $\hat{\boldsymbol{x}}$ is a size n column vector, $\hat{\boldsymbol{A}}$ is a $m \times n$ integer matrix and $\boldsymbol{b}$ is a size m integer column vector. Adding positive slack variables to change the constraints to equalities yields (5.2) where $\boldsymbol{c} = \left[\hat{\boldsymbol{c}} \mid \boldsymbol{0}\right]$, $\boldsymbol{x}' = \left[\hat{\boldsymbol{x}}' \mid \boldsymbol{s}'\right]$, and $\boldsymbol{A} = \left[\hat{\boldsymbol{A}} \mid \boldsymbol{I}\right]$.

$$\begin{aligned} \text{Max} \quad & \boldsymbol{cx} \\ \text{s.t.} \quad & \boldsymbol{Ax} = \boldsymbol{b} \\ & \boldsymbol{x} \ge \boldsymbol{0} \text{ and integer} \end{aligned} \tag{5.2}$$

Let $\boldsymbol{B}$ be the optimal basis from the LP relaxation and $\boldsymbol{N}$ be the matrix containing the non-basic columns, separating $\boldsymbol{A}$ into $\boldsymbol{B}$ and $\boldsymbol{N}$ yields (5.3) where $\boldsymbol{c_B}$

and $\boldsymbol{x_B}$ are the cost coefficients and variables associated with the basic columns and $\boldsymbol{c_N}$ and $\boldsymbol{x_N}$ are those associated with the non-basic columns.

$$\begin{aligned} \text{Max} \quad & \boldsymbol{c_B x_B + c_N x_N} \\ \text{s.t.} \quad & \boldsymbol{B x_B + N x_N = b} \\ & \boldsymbol{x_B \geq 0} \text{ and integer} \\ & \boldsymbol{x_N \geq 0} \text{ and integer} \end{aligned} \tag{5.3}$$

Solving for $\boldsymbol{x_B}$ in terms of $\boldsymbol{x_N}$ yields

$$\boldsymbol{x_B = B^{-1}\left(b - N x_N\right) = B^{-1} b - B^{-1} N x_N} \tag{5.4}$$

Substituting for $\boldsymbol{x_B}$ using Equation 5.4 yields

$$\begin{aligned} \text{Max} \quad & \boldsymbol{c_B B^{-1} b - \left(c_B B^{-1} N - c_N\right) x_N} \\ \text{s.t.} \quad & \boldsymbol{x_B = B^{-1} b - B^{-1} N x_N} \\ & \boldsymbol{x_B \geq 0} \text{ and integer} \\ & \boldsymbol{x_N \geq 0} \text{ and integer} \end{aligned} \tag{5.5}$$

The reduced costs from the optimal LP solution are defined as $\boldsymbol{\bar{c}_N = \left(c_B B^{-1} N - c_N\right)}$. Since we are maximizing and the variables do not have upper bounds, $\boldsymbol{\bar{c}_N \geq 0}$ at optimality and the objective function value is $\boldsymbol{z^*_{\text{LP}} = c_B B^{-1} b}$. Therefore, the objective function in (5.5) can be rewritten as $\boldsymbol{z = z^*_{\text{LP}} - \bar{c}_N x_N}$ and the problem can be restated as

$$\begin{aligned} \text{Min} \quad & \boldsymbol{\bar{c}_N x_N} \\ \text{s.t.} \quad & \boldsymbol{x_B = B^{-1} b - B^{-1} N x_N} \\ & \boldsymbol{x_B \geq 0} \text{ and integer} \\ & \boldsymbol{x_N \geq 0} \text{ and integer} \end{aligned} \tag{5.6}$$

The basic variables, $\boldsymbol{x_B}$, will be integer if they satisfy the congruence relationship $\boldsymbol{B^{-1}b} - \boldsymbol{B^{-1}Nx_N} \equiv 0 \pmod 1$ which is equivalent to $\boldsymbol{B^{-1}Nx_N} \equiv \boldsymbol{B^{-1}b} \pmod 1$. Substituting in this constraint and *relaxing* the non-negativity requirement on $\boldsymbol{x_B}$ yields

$$\begin{aligned} \text{Min} \quad & \bar{\boldsymbol{c}}_N \boldsymbol{x}_N \\ \text{s.t.} \quad & \boldsymbol{B^{-1}Nx_N} \equiv \boldsymbol{B^{-1}b} \,(\mathbf{mod\ 1}) \\ & \boldsymbol{x_N} \geq \mathbf{0} \text{ and integer} \end{aligned} \tag{5.7}$$

Since the constraint in (5.7) is based on modulo 1, all that is needed from the columns $\boldsymbol{B^{-1}N}$ and $\boldsymbol{B^{-1}b}$ are the fractional parts, $f_{ij} = \boldsymbol{B^{-1}N}_{ij} - \lfloor \boldsymbol{B^{-1}N}_{ij} \rfloor$ and $f_i = \boldsymbol{B^{-1}b}_i - \lfloor \boldsymbol{B^{-1}b}_i \rfloor$. For example, the fractional part of 3.25 is 0.25 and the fractional part of -3.25 is 0.75. Let $\boldsymbol{\alpha}_j$ be a column vector containing the fractional parts of $\boldsymbol{B^{-1}N}_j$ and $\boldsymbol{\alpha}_0$ be a column vector containing the fractional parts of $\boldsymbol{B^{-1}b}$. The resulting GMP from (5.1) is

$$\begin{aligned} \text{Min} \quad & \bar{\boldsymbol{c}}_N \boldsymbol{x}_N \\ \text{s.t.} \quad & \textstyle\sum_{j=1}^{n} \boldsymbol{\alpha}_j x_j \equiv \boldsymbol{\alpha}_0 \,(\mathbf{mod\ 1}) \\ & \boldsymbol{x_N} \geq \mathbf{0} \text{ and integer} \end{aligned} \tag{5.8 - GMP}$$

Example 5.1

Consider the following integer linear program (ILP) (Jensen and Bard 2003).

$$\begin{aligned} \text{Max} \quad & -2x_1 + x_2 \\ \text{s.t.} \quad & 9x_1 - 3x_2 \geq 11 \\ & x_1 + 2x_2 \leq 10 \\ & 2x_1 - x_2 \leq 7 \\ & x_1, x_2 \geq 0 \text{ and integer} \end{aligned} \tag{P1}$$

The feasible region is shown in Figure 5.1. The optimal solution to the LP relaxation is $x_1 = 52/21$, $x_2 = 79/21$, and $z = -25/21$. The dashed lines are the isovalue lines of z and the shaded area is the feasible region for P1 with optimal solution $x_1 = 2$, $x_2 = 2$, and $z = -2$, which is *not* the closet integer point to $\boldsymbol{x}^*_{\text{LP}}$ in Euclidean distance. It is the first integer point encountered as we move the isovalue lines of z into the feasible region.

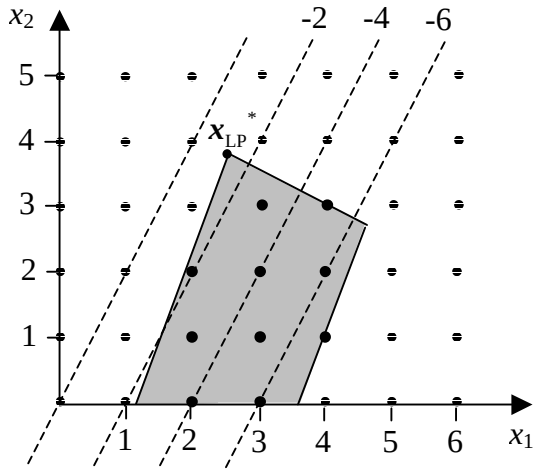

Figure 5.1 – feasible region in $\boldsymbol{X}$ space

Adding slacks yields

$$
\begin{aligned}
\text{Max} \quad & -2x_1 + x_2 \\
\text{s.t.} \quad & 9x_1 - 3x_2 - s_1 = 11 \\
& x_1 + 2x_2 + s_2 = 10 \\
& 2x_1 - x_2 + s_3 = 7 \\
& x_1, x_2, s_1, s_2, s_3 \geq 0 \text{ and integer}
\end{aligned}
$$

At $\boldsymbol{x}^*_{\text{LP}}$ the basic variables are x_1, x_2, and s_3 with

$$\boldsymbol{B}=\begin{bmatrix}9 & -3 & 0\\ 1 & 2 & 0\\ 2 & -1 & 1\end{bmatrix} \quad \boldsymbol{B}^{-1}=\begin{bmatrix}2/21 & 3/21 & 0\\ -1/21 & 9/21 & 0\\ -5/21 & 3/21 & 1\end{bmatrix} \quad \boldsymbol{B}^{-1}\boldsymbol{b}=\begin{bmatrix}52/21\\ 79/21\\ 122/21\end{bmatrix} \quad \boldsymbol{B}^{-1}\boldsymbol{N}=\begin{bmatrix}-2/21 & 3/21\\ 1/21 & 9/21\\ 5/21 & 3/21\end{bmatrix}$$

The reduced costs for s_1 and s_2 are 5/21 and 3/21 respectively. The resulting GMP is

$$\begin{aligned}
\text{Min}\quad & 5/21s_1+3/21s_2 \\
\text{s.t.}\quad & 19/21s_1+3/21s_2 \equiv 10/21 \pmod 1 \\
& 1/21s_1+9/21s_2 \equiv 16/21 \pmod 1 \\
& 5/21s_1+3/21s_2 \equiv 17/21 \pmod 1 \\
& s_1, s_2 \geq 0 \text{ and integer}
\end{aligned} \qquad \text{(GMP1)}$$

Note element α_{11} = 19/21 instead of -2/21 based on the previous definition of fractional part.

5.1.2 The Fractional Group

The problem defined in (5.8) is called the *group* minimization problem because the vectors $\boldsymbol{\alpha}_j$ and $\boldsymbol{\alpha}_0$ are elements of a finite abelian group under component-wise addition modulo 1. The group equation is the congruence relationship

$$\sum_{j=1}^{n} \boldsymbol{\alpha}_j x_j \equiv \boldsymbol{\alpha}_0 \ (\mathbf{mod\ 1}) \qquad (5.9)$$

Let D be the *absolute value* of the determinant of the basis $\boldsymbol{B}$. By Cramer's rule, the form of each vector element α_{ij} is I_{ij}/D where I_{ij} is integer and $I_{ij} < D$. Since each element of the vector can take on any one of D distinct values, the size of the *full* group is D^m where m is the number of rows in the problem (size of the vector). However, the elements $\boldsymbol{\alpha}_j$ generate a subgroup of size $\leq D$ (Johnson 1980,

Salkin and Mathur 1989). Let $H(\boldsymbol{\alpha}) = \langle \boldsymbol{\alpha}_j \rangle$ be the subgroup generated by the columns of the GMP.

Example 5.2

For P1 we have

$$\boldsymbol{\alpha}_1 = g_1 = \begin{bmatrix} 19/21 \\ 1/21 \\ 5/21 \end{bmatrix}$$

The I_{ij} in element g_1 are all relatively prime with $D = 21$, therefore g_1 generates the 21 element cyclic subgroup, $H(\boldsymbol{\alpha})$ (Salkin and Mathur 1989). ($\boldsymbol{\alpha}_2$ generates a cyclic subgroup of order 7 which is subsumed in $H(\boldsymbol{\alpha})$.) The elements of $H(\boldsymbol{\alpha})$ are shown below.

g_1	g_2	g_3	g_4	g_5	g_6	g_7
$\begin{bmatrix} 19/21 \\ 1/21 \\ 5/21 \end{bmatrix}$	$\begin{bmatrix} 17/21 \\ 2/21 \\ 10/21 \end{bmatrix}$	$\begin{bmatrix} 15/21 \\ 3/21 \\ 15/21 \end{bmatrix}$	$\begin{bmatrix} 13/21 \\ 4/21 \\ 20/21 \end{bmatrix}$	$\begin{bmatrix} 11/21 \\ 5/21 \\ 4/21 \end{bmatrix}$	$\begin{bmatrix} 9/21 \\ 6/21 \\ 9/21 \end{bmatrix}$	$\begin{bmatrix} 7/21 \\ 7/21 \\ 14/21 \end{bmatrix}$
g_8	g_9	g_{10}	g_{11}	g_{12}	g_{13}	g_{14}
$\begin{bmatrix} 5/21 \\ 8/21 \\ 19/21 \end{bmatrix}$	$\begin{bmatrix} 3/21 \\ 9/21 \\ 3/21 \end{bmatrix}$	$\begin{bmatrix} 1/21 \\ 10/21 \\ 8/21 \end{bmatrix}$	$\begin{bmatrix} 20/21 \\ 11/21 \\ 13/21 \end{bmatrix}$	$\begin{bmatrix} 18/21 \\ 12/21 \\ 18/21 \end{bmatrix}$	$\begin{bmatrix} 16/21 \\ 13/21 \\ 2/21 \end{bmatrix}$	$\begin{bmatrix} 14/21 \\ 14/21 \\ 7/21 \end{bmatrix}$
g_{15}	g_{16}	g_{17}	g_{18}	g_{19}	g_{20}	g_{21}
$\begin{bmatrix} 12/21 \\ 15/21 \\ 12/21 \end{bmatrix}$	$\begin{bmatrix} 10/21 \\ 16/21 \\ 17/21 \end{bmatrix}$	$\begin{bmatrix} 8/21 \\ 17/21 \\ 1/21 \end{bmatrix}$	$\begin{bmatrix} 6/21 \\ 18/21 \\ 6/21 \end{bmatrix}$	$\begin{bmatrix} 4/21 \\ 19/21 \\ 11/21 \end{bmatrix}$	$\begin{bmatrix} 2/21 \\ 20/21 \\ 16/21 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

We have $\boldsymbol{\alpha}_1 = g_1$, and $\boldsymbol{\alpha}_2 = g_1^9 = g_{1*}9 = g_9$. The I_{ij} in element g_9 share a common divisor with D, namely 3, so the order of element g_9 is 21/3 = 7. Since $\boldsymbol{\alpha}_0 = g_1 16 = 16 g_1 = g_{16}$, setting $s_1 = 16$ and $s_2 = 0$ satisfies the GMP1 constraints. The corresponding $x_{\boldsymbol{Bj}}$ (from Equation 5.4) are $x_1 = 4$, $x_2 = 3$, $s_3 = 2$. This yields a feasible but *not* optimal solution for P1 with $z = -5$. A lower cost solution is generated by $\alpha_0 = g_{16} = g_1 + g_{15} = g_1 + 4g_9$ which implies that $s_1 = 1$ and $s_2 = 4$ and, further, that $x_1 = 2$, $x_2 = 2$, $s_3 = 5$. This yields the optimal solution for P1 with $z = -2$.

Incidentally, the relaxations of the rows in (5.8) from $\equiv(\mathbf{mod1})$ to $\geq$ are the Gomory fractional cuts. These row vectors also generate a finite abelian group under component-wise addition modulo 1 that has a size of $\leq D$. The group generated by the rows and the group generated by the columns are isomorphic (Gomory 1963). Gomory also notes the group is often cyclic and therefore can be generated by a single element.

5.1.3 The Sufficient Condition for $\boldsymbol{x_B \geq 0}$

The GMP attempts to find the integer point with the minimum *weighted* (by $\bar{\boldsymbol{c}}$) Euclidean distance to $\boldsymbol{x}^*_{\mathrm{LP}}$. As illustrated in Figure 5.1, the optimal solution to an ILP need not be the closest integer point to $\boldsymbol{x}^*_{\mathrm{LP}}$ in *unweighted* Euclidean distance. We can solve the GMP, given in 5.8, for any ILP and when $\boldsymbol{x_N}^*$ is found, $\boldsymbol{x_B}^*$ can be derived using Equation 5.4. If $\boldsymbol{x_B}^* \geq \mathbf{0}$ then $\boldsymbol{x}^* = \left[\boldsymbol{x_B}^* \middle| \boldsymbol{x_N}^* \right]$ is the optimal solution to the original ILP.

Gomory (1965, 1969) provides a sufficient condition under which $\boldsymbol{x_B}$ will be non-negative. From the LP, we have $\boldsymbol{Bx_B} = \boldsymbol{b}$ and $\boldsymbol{x_B} \geq \boldsymbol{0}$, which implies that $\boldsymbol{b}$ is contained in the cone generated by the columns of $\boldsymbol{B}$, $K(\boldsymbol{B})$. Solving the GMP implies an associated $\boldsymbol{x_B}$. Since $\boldsymbol{Bx_B} = \boldsymbol{b} - \boldsymbol{Nx_N}$, if $\boldsymbol{b} - \boldsymbol{Nx_N}$ is contained in $K(\boldsymbol{B})$, $\boldsymbol{x_B} \geq \boldsymbol{0}$.

If $\boldsymbol{b}$ is on the surface of $K(\boldsymbol{B})$, the solution is degenerate and at least one of the basic variables is equal to 0. Since we are perturbing b_i by $(Nx_N)_i$, we need to have $\boldsymbol{b}$ a sufficient distance from the perimeter of $K(\boldsymbol{B})$ to ensure the optimal solution to the GMP yields $\boldsymbol{x_B} \geq \boldsymbol{0}$. In other words, the basic variable values from the LP optimal must be sufficiently greater than zero. If the LP yields a degenerate solution, then $\boldsymbol{b}$ is on the boundary of $K(\boldsymbol{B})$. Based on this observation, Balas (1973) shows the sufficient condition is never satisfied for binary programming problems.

5.1.4 Column Reduction

Gomory (1969) provides two cases for eliminating a column from the GMP. From a *strictly* GMP perspective, if all of the elements in a non-basic column are integer, that redundant column will map to the group identity (zero vector) and contribute nothing to forcing the basic variables to be integer. Since $\bar{\boldsymbol{c}}_{\boldsymbol{N}} \geq \boldsymbol{0}$, an all integer non-basic column will never appear in the optimal solution to the GMP and may be eliminated. Similarly, if two or more non-basic columns have the same fractional components, they map to the same group element. Again, from a strict

GMP perspective, the column with the smallest $\bar{c}_j$ can be retained and the other *redundant* columns may be discarded.

Unfortunately these redundant columns *might* be needed to construct an optimal solution to the original ILP. If the optimal GMP solution yields a negative $\boldsymbol{x_B}$, the more costly "redundant" columns may be needed in the ILP to ensure $\boldsymbol{x_B} \geq \boldsymbol{0}$.

For example, we could have two columns, $\boldsymbol{N}_j$ and $\boldsymbol{N}_k$, with identical fractional components. If $\bar{c}_j \geq \bar{c}_k$ and $\|\boldsymbol{N}_j\| \leq \|\boldsymbol{N}_k\|$, it may be necessary to absorb the additional cost of $\bar{c}_j$ so that $\boldsymbol{x_B} \geq \boldsymbol{0}$. Additionally, if we have $\boldsymbol{x_{Bi}} < 0$ and an all-integer $\boldsymbol{N}_j$ with $\boldsymbol{B^{-1}N}_{ij} < 0$ we may need to use $\boldsymbol{N}_j$ to ensure $\boldsymbol{x_B} \geq \boldsymbol{0}$.

Example 5.3

Assume the solution to the LP relaxation generated the following

$$\bar{\boldsymbol{c}}_N = \begin{bmatrix} 2/21 & 3/21 & 2/21 \end{bmatrix} \quad \boldsymbol{B^{-1}b} = \begin{bmatrix} 50/21 \\ 110/21 \\ 3/21 \end{bmatrix} \quad \boldsymbol{B^{-1}N} = \begin{bmatrix} 21/21 & 8/21 & 29/21 \\ 42/21 & 5/21 & 26/21 \\ -42/21 & 24/21 & 66/21 \end{bmatrix}$$

The associated GMP is

$$\begin{aligned} \text{Min} \quad & 2/21 x_{N1} + 3/21 x_{N2} + 2/21 x_{N3} \\ \text{s.t.} \quad & 0x_{N1} + 8/21 x_{N2} + 8/21 x_{N3} \equiv 8/21 \pmod 1 \\ & 0x_{N1} + 5/21 x_{N2} + 5/21 x_{N3} \equiv 5/21 \pmod 1 \\ & 0x_{N1} + 3/21 x_{N2} + 3/21 x_{N3} \equiv 3/21 \pmod 1 \\ & x_{N1}, x_{N2}, x_{N3} \geq 0 \text{ and integer} \end{aligned}$$

The first column has all integer values and maps to the group identity. The second and third columns have the identical fractional components and $\bar{c}_{N2} > \bar{c}_{N3}$. So the first and second columns can be eliminated from the GMP.

Cleary the optimal solution to the GMP is $x_{N1} = 0$, $x_{N2} = 0$, and $x_{N3} = 1$ with a cost of 2. However, this GMP solution yields $x_{B1} = 1$, $x_{B2} = 4$, and $x_{B3} = -3$ which is an infeasible solution to ILP. The minimum cost solution for the GMP that yields $\boldsymbol{x_B} \geq \mathbf{0}$ is $x_{N1} = 1$, $x_{N2} = 1$, and $x_{N3} = 0$ with cost 5. The resulting basic variables are $x_{B1} = 1$, $x_{B2} = 3$, and $x_{B3} = 1$.

5.1.5 The Factor Group Minimization Problem (FGMP)

An alternate form of the GMP exists based on a factor group $M(\boldsymbol{I})/M(\boldsymbol{B})$ where $M(\boldsymbol{I})$ is the group of all integer vectors of size m and $M(\boldsymbol{B})$ is the subgroup of all linear integer combinations of the columns in the optimal LP basis. The size of the factor group is $\leq D$ and the group is isomorphic to $H(\boldsymbol{\alpha})$, therefore, the solutions to both versions are identical (Gomory 1969, Salkin and Mathur 1989).

The FGMP formulation is found by finding the row and column operations required to translate $\boldsymbol{B}$ to the Smith-Normal form and performing those same row and column operations on $\boldsymbol{N}$ and $\boldsymbol{b}$. If the method used to solve the GMP relies on the full group not just the subgroup, as many early approaches did (Glover 1968, Salkin and Mathur 1989, Wolsey 1971), it may be beneficial to use the FGMP formulation (Johnson 1980).

5.1.6 Another View of the GMP

We can view the GMP generically in group theoretic terms. Given some arbitrary group G we want to find the minimum cost combination of group elements that sum to some group element g_0.

$$\begin{aligned} \text{Min} \quad & \sum_{g \in G} c(g)t(g) \\ \text{s.t.} \quad & \sum_{g \in G} gt(g) = g_0 \\ & t(g) \geq 0 \text{ and integer } \forall\ g \in G \end{aligned} \tag{5.10}$$

where $c(g)$ is the real-valued weight or cost for group element g and $t(g)$ is the power or multiplier for element g (Gomory 1969).

5.2 Corner Polyhedra

In this section, we explore the geometry behind the GMP. Gomory (1967, 1969) defines corner polyhedra as the main area of interest in an ILP and the feasible region for the GMP.

5.2.1 Corner Polyhedra in *X* Space

Let $\boldsymbol{X}$ represent the original decision variables of the ILP (excluding the slack variables). The corner polyhedron in $\boldsymbol{X}$ space ($CP^{\boldsymbol{X}}$) is found by first relaxing the bounds on the basic variables from the optimal solution to the LP relaxation (Gomory 1967, 1969). For each $x_j \in \boldsymbol{X}$, if variable x_j is basic then $x_j > 0$ and the hyperplane $x_j = 0$ does not pass through $\boldsymbol{x}_{\text{LP}}^*$. Similarly, if a slack variable s_j is basic then $s_j > 0$ and $\boldsymbol{a}_i\boldsymbol{x} < b_i$, where $\boldsymbol{a}_i$ is the ith row of $\boldsymbol{A}$, and the hyperplane $\boldsymbol{a}_i\boldsymbol{x} = b_i$ also does not

pass through $\boldsymbol{x}_{\mathrm{LP}}^{*}$. So relaxing the lower bounds on the basic variables is equivalent to relaxing all non-binding constraints at $\boldsymbol{x}_{\mathrm{LP}}^{*}$.

We are left with only the hyperplanes passing through $\boldsymbol{x}_{\mathrm{LP}}^{*}$ which form an unbounded polyhedral cone *K*. All of the feasible integer solutions to the original ILP are contained in this cone. The corner polyhedron, $CP^{\boldsymbol{X}}$, is the convex hull of the integer points in *K* (i.e. $CP^{\boldsymbol{X}}$ = conv(*K*)). The optimal solution to the ILP is a vertex of $CP^{\boldsymbol{X}}$ which will often have significantly less vertices than the original feasible region conv(*P*) (Gomory 1969).

Example 5.4

For P1 at $\boldsymbol{x}_{\mathrm{LP}}^{*}$, we have x_1, x_2, and s_3 basic. Relaxing the lower bounds on these variables yields unbounded polyhedral cone *K* shown with solid lines in Figure 5.2. The associated convex hull is the corner polyhedron shown as the shaded area in Figure 5.2. The ten original feasible points are labeled A through J. C is the optimal ILP solution.

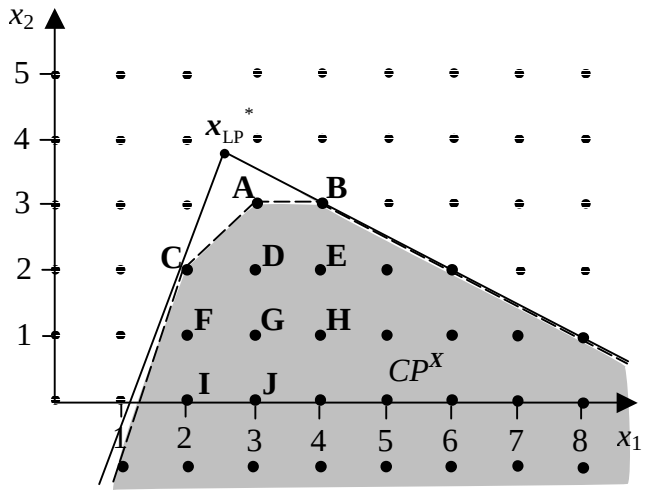

Figure 5.2 – corner polyhedron in $\boldsymbol{x}$ space, $CP^{\boldsymbol{X}}$

5.2.2 Corner Polyhedra in $\boldsymbol{X_N}$ Space

Let $\boldsymbol{X_N}$ represent the non-basic variables from the optimal solution to the LP relaxation, the variables in the GMP (including non-basic slack variables). We can also construct the corner polyhedron in $\boldsymbol{X_N}$ space ($CP^{\boldsymbol{X_N}}$) (Gomory 1969). To do so we relax the congruence relationship (5.9) to

$$\sum_{j=1}^{n} \alpha_{ij} x_j \geq \alpha_{i0} \quad \forall i = 1,..,m \tag{5.11}$$

Certainly any feasible point for (5.9) satisfies the set of equations in (5.11). The equations in (5.11) are the classical Gomory fractional cuts. Let $P^{\boldsymbol{X_N}}$ be the unbounded convex polyhedron formed by (5.11) and the non-negativity of $\boldsymbol{x_N}$. The convex hull of the integer points in $P^{\boldsymbol{X_N}}$ that satisfy (5.9) is the corner polyhedron in

$\boldsymbol{X_N}$ space, CP^{X_N}. Any vertex v of CP^{X} corresponds to a vertex of CP^{X_N} and vice-versa, so the optimal solution to the original ILP is also a vertex of CP^{X_N} (Gomory 1969).

Example 5.5

For P1 at $\boldsymbol{x}_{\mathrm{LP}}^{*}$, we have s_1 and s_2 non-basic. Relaxing the constraints yields the following constraints:

$$19/21 s_1 + 3/21 s_2 \geq 10/21$$
$$1/21 s_1 + 9/21 s_2 \geq 16/21$$
$$5/21 s_1 + 3/21 s_2 \geq 17/21$$

The convex hull satisfying these constraints yields the corner polyhedron in $\boldsymbol{X_N}$ space as shown as the shaded area in Figure 5.3.

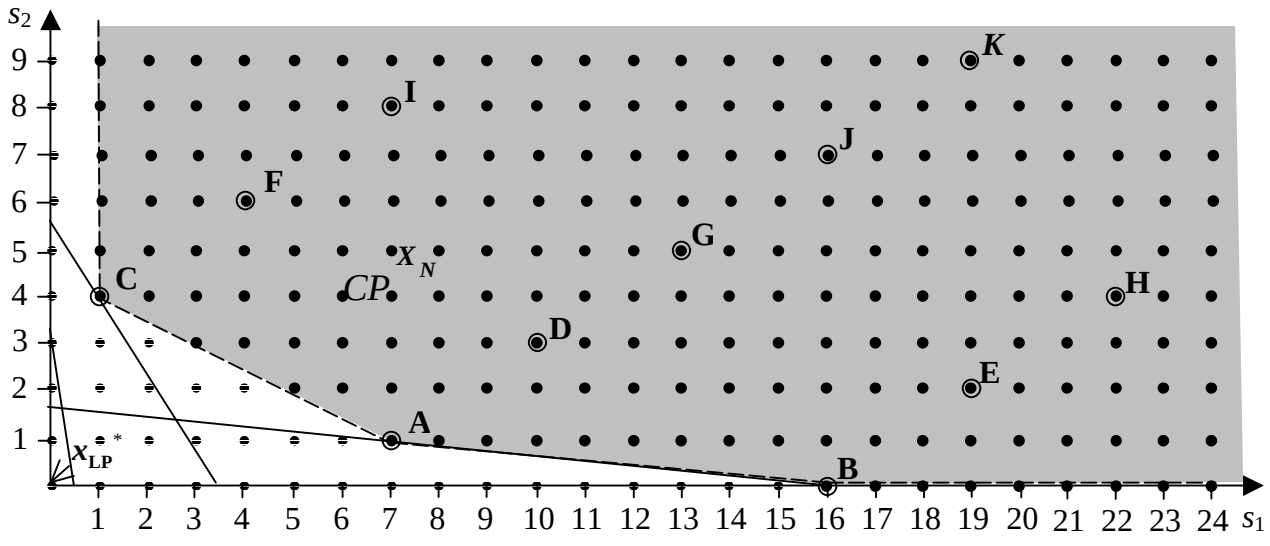

Figure 5.3 – corner polyhedron in $\boldsymbol{X_N}$ space, CP^{X_N}

The reduced cost of each non-basic variable represents the cost of moving one unit away from the origin in $\boldsymbol{X_N}$ space along that variable's axis. Of course, not every integer point in $\boldsymbol{X_N}$ space yields integer basic variables when $\boldsymbol{x_B}$ is calculated using Equation 5.4, only those points satisfying (5.9). These *integer-basis* points are circled in Figure 5.3 and labeled based on their corresponding point in $\boldsymbol{X}$ space. For instance, point K is an integer-basis point and is GMP1 feasible but it is infeasible for P1 as it yields $\boldsymbol{x_B} < 0$. The integer-basis points comprise a *sub-lattice* on the integer lattice of $CP^{\boldsymbol{X_N}}$. Of course this sub-lattice is not random; it follows a distinct pattern based on the group $H(\boldsymbol{\alpha})$. The density of these points within the integer lattice of $CP^{\boldsymbol{X_N}}$ is related to the order of the group elements associated with each non-basic variable.

Theorem 5.1

The frequency of integer-basis points along axis $\boldsymbol{k}$ of the integer lattice is order($\boldsymbol{\alpha}_k$).

Proof

Let $\hat{\boldsymbol{x}}_{\boldsymbol{N}}$ be an integer-basis point in $\boldsymbol{X_N}$ space. Fix $x_j = \hat{x}_j,\ \forall x_j \in \boldsymbol{X}_{\boldsymbol{N}}$ then choose some $x_k \in \boldsymbol{X}_{\boldsymbol{N}}$ to become free. So we have one degree of freedom and are exploring integer lattice points along k^{th} axis. From the congruence relationship (5.9) we know the point is an integer-basis point iff

$$\boldsymbol{\alpha}_k x_k \equiv \boldsymbol{\alpha}_0 - \sum_{\substack{j \in \boldsymbol{N} \\ j \neq k}} \boldsymbol{\alpha}_j \hat{x}_j \equiv \boldsymbol{\alpha}^* \left(\text{mod } 1\right).$$

We know $x_k = \hat{x}_k$ satisfies the above relationship and therefore $\boldsymbol{\alpha}^*$ is a power of $\boldsymbol{\alpha}_k$. If $\boldsymbol{\alpha}_k \hat{x}_k \equiv \boldsymbol{\alpha}^* \left(\text{mod } 1\right)$, then we must have $\boldsymbol{\alpha}_k \left(\hat{x}_k + p \text{ order}\left(\boldsymbol{\alpha}_k\right)\right) \equiv \boldsymbol{\alpha}^* \left(\text{mod } 1\right)$ for all

integer p by the definition of the order of a group element. Therefore the frequency of integer-basis points along axis k of the integer lattice is order($\boldsymbol{\alpha}_k$). □

As noted earlier, the form of the elements of $\boldsymbol{\alpha}_j$ is I_{ij}/D where I_{ij} is integer and $I_{ij} < D$. Let gcd(D,I_j) be the greatest common divisor of D and the I_{ij} over j. The order of element $\boldsymbol{\alpha}_j$ is D/gcd(D,I_j) (Salkin and Mathur 1989).

Example 5.6

In problem P1, we have order($\boldsymbol{\alpha}_1$) = 21 and order($\boldsymbol{\alpha}_2$) = 7. In Figure 5.3, we can clearly see that integer-basis points are 21 lattice points apart along the s_1 axis and 7 lattice points along the s_2 axis.

The order of any element in a group must be less than the size of the group. Since the size of $H(\boldsymbol{\alpha})$ is $\leq D$, D gives us an upper bound on the frequency of integer-basis points along any axis. It is well known that if D = 1, the LP relaxation will yield an all integer basis. In GMP terms, D = 1 implies $\|H(\boldsymbol{\alpha})\| = 1$ and every integer point in $\boldsymbol{X_N}$ space is an integer-basis point.

If $\boldsymbol{B}^{-1}\boldsymbol{b}$ is integer then $\boldsymbol{\alpha}_0$ in (5.8) is the group identity or zero vector. In this case, $\boldsymbol{x_N}$ = 0 is a feasible solution to the GMP and the origin in $\boldsymbol{X_N}$ space is on the integer-basis sub-lattice and inside the corner polyhedron. Since the group identity must be a power of every group element, there are integer-basis sub-lattice points on each axis in $\boldsymbol{X_N}$ space. So the corner polyhedron becomes the positive quadrant and the origin its only vertex.

5.3 Local Search Neighborhoods in X_N Space

We can solve the GMP by searching the corner polyhedron in $\boldsymbol{X_N}$ space. One simple neighborhood is to increment or decrement by 1 a single non-basic variable $\boldsymbol{x_{Nj}}$ with cost $\pm\bar{c}_j$ for j = 1 to n. This neighborhood would traverse the integer lattice in $\boldsymbol{X_N}$ but would require a penalty function to drive the search to the sub-lattice points yielding an integer basis. While this neighborhood is simple to implement and allows us to take advantage of the fine grain gradient provided by the individual $\bar{c}_j$, it is affected by the size of D. Experimentation has shown it to be ineffective for all but very small problems. The distance between the points on the sub-lattice is too great and as the problems get larger the algorithm failed to find a single sub-lattice point. So let us consider how to traverse the integer-basis sub-lattice.

5.3.1 Identity Moves

Let $\hat{\boldsymbol{x}}_N$ and $\hat{\boldsymbol{y}}_N$ be integer-basis points. Since both satisfy the congruence relationship (5.9), define $\boldsymbol{d} = \hat{\boldsymbol{x}}_N - \hat{\boldsymbol{y}}_N$, so $\sum_{j=1}^{n} \boldsymbol{\alpha}_j d_j \equiv \mathbf{0} \left(\mathbf{mod}\, \mathbf{1}\right)$ with $\mathbf{0}$ being the group identity. Note that $\hat{\boldsymbol{x}}_N + p\, \boldsymbol{d}$ is an integer-basis point for all integer p. The n-dimensional vector $\boldsymbol{d}$ allows us to move from one integer-basis point to another in direction of $\boldsymbol{d}$; it captures the direction and the step size.

The move value $\bar{\boldsymbol{c}}_N \boldsymbol{d}$ measures the change in weighted Euclidean distance to $\boldsymbol{x}_{\mathrm{LP}}^{*}$ and is *independent of the current point.* Given a known set of identity moves, we can order them by move value from best to worst. At every iteration we take the

first available move based on our search parameters and avoid searching the entire neighborhood.

Example 5.7

In Figure 5.4, G_N = (13, 5) with z_{GMP} = 80/21 and J_N = (16, 7) with z_{GMP} = 101/21. Letting $\boldsymbol{d} = J_N - G_N$ = (3, 2), then $\bar{\boldsymbol{c}}_N\boldsymbol{d}$ = 21/21. Moving along this line we have $G_N - \boldsymbol{d}$ = (10, 3) = D_N with z_{GMP} = 59/21 and $G_N - 2\boldsymbol{d}$ = (7, 1) = A_N with z_{GMP} = 38/21. Calculating $\boldsymbol{x_B}$ using Equation 5.4 yields solutions D and A from Figure 5.2 with z_{IP} = -4 and z_{IP} = -3 respectively.

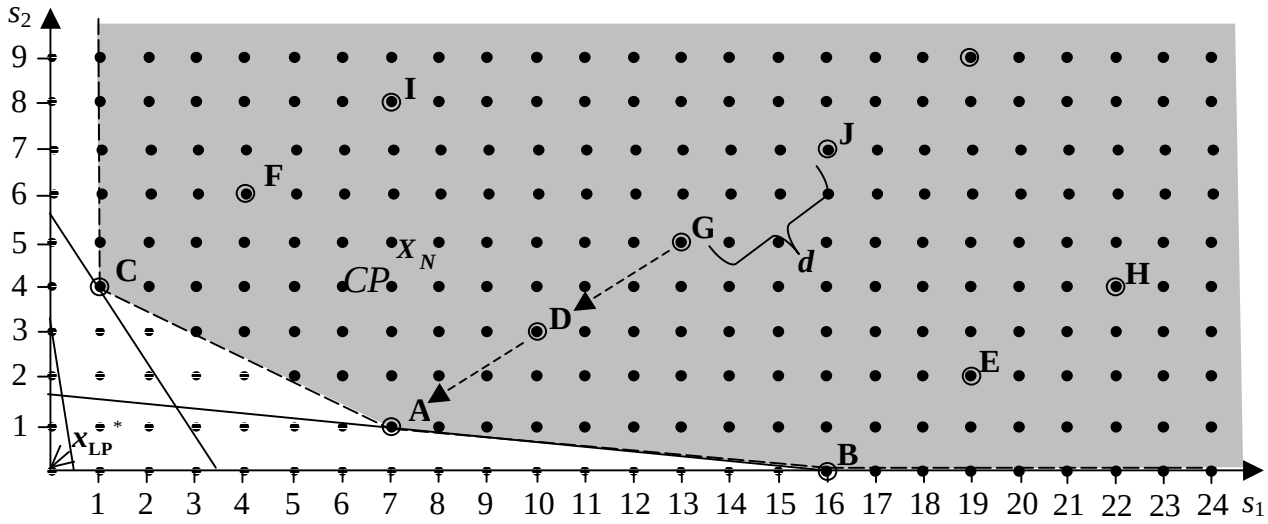

Figure 5.4 – identity moves in $\boldsymbol{X_N}$ space

Deriving identity moves as the difference between two solutions does not require either solution to be feasible. The solutions of course must be integer. However, they may have some $\boldsymbol{x_B} < \boldsymbol{0}$ and/or $\boldsymbol{x_N} < \boldsymbol{0}$. Since we calculate $\boldsymbol{x_B}$ from Equation 5.4 will have always have $\boldsymbol{Ax} = \boldsymbol{b}$. The difference between two infeasible

moves still gives us a valid identity move and we may be able to use such moves to guide the search into the feasible region.

Example 5.8

Although solutions $\boldsymbol{x} = (0, 0, -11, 10, 7)^t$ and $\boldsymbol{y} = (0, 1, -14, 8, 8)^t$ satisfy $\boldsymbol{Ax = b}$ from P1, they are infeasible because both have $s_1 < 0$. The associated non-basic variable vectors are $\boldsymbol{x_N} = (-11, 10)$ and $\boldsymbol{y_N} = (-14, 8)$. Letting $\boldsymbol{d} = \boldsymbol{x_N} - \boldsymbol{y_N} = (3, 2)$ gives us the same identity move found earlier. Unfortunately, since the vector connecting $\boldsymbol{x_N}$ and $\boldsymbol{y_N}$ does not intersect the P1 feasible region, we cannot reach the feasible region using just this move.

5.3.2 Generating Identity Moves

Generating identity moves as the difference between integer solutions can be difficult if integer solutions are hard to come by. Fortunately, these solutions need not be feasible. We now define a set of easily generated *atomic* identity moves. As Theorem 5.3 will show, all possible identity moves can be expressed as a linear integer combination of these atomic moves.

Using $x_j = 0 \ \forall x_j \in \boldsymbol{X}$ as our baseline solution, we can create additional solutions by setting each x_j to 1 in turn and adjusting the slack variables. We then subtract the difference between each new solution and the baseline solution (including corresponding slack variables). The resulting difference vector details the change in each non-basic variable *and* each basic variable as a result of the move.

The value of the move can be calculated by multiplying the vector of reduced costs by the difference vector since $\bar{c}_j = 0$ if x_j is basic.

Here is the procedure that builds atomic identity move j (denoted d^j) for $j \le n$:

1. Set $\nabla x_j = 1,\ \nabla x_k = 0\ \forall x_k \in \boldsymbol{X}, k \neq j$

2. For each slack variable s_i

$$\nabla s_i = \begin{cases} -\hat{a}_{ij} & \text{if constraint } i \text{ is } \le \\ \hat{a}_{ij} & \text{if constraint } i \text{ is } \ge \end{cases} \quad \text{where } \hat{a}_{ij} \text{ is from } \hat{A} \text{ in Equation 5.1}$$

3. $\boldsymbol{d}^j = \left[\nabla \boldsymbol{x} \middle| \nabla \boldsymbol{s}\right]^t$ has cost $\bar{\boldsymbol{c}}\boldsymbol{d}^j$

Example 5.9

For P1, build $\boldsymbol{d}^1$ from $\nabla x_1 = 1,\ \nabla x_2 = 0,\ \nabla s_1 = 9,\ \nabla s_2 = -1$, and $\nabla s_3 = -2$ so $\boldsymbol{d}^1 = (1, 0, 9, -1, -2)^t$. The reduced cost vector is $\bar{\boldsymbol{c}} = (0, 0, 5/21, 3/21, 0)$ so $\bar{\boldsymbol{c}}\boldsymbol{d}^1 = 2$. For $\boldsymbol{d}^2$, use $\nabla x_1 = 0,\ \nabla x_2 = 1,\ \nabla s_1 = -3,\ \nabla s_2 = -2$, and $\nabla s_3 = 1$ so $\boldsymbol{d}^2 = (0, 1, -3, -2, 1)^t$ and $\bar{\boldsymbol{c}}\boldsymbol{d}^2 = -1$. Starting from any integer-basis point in $\boldsymbol{X}^N$ space we can add linear-integer combinations of these moves and reach another integer-basis point. Going back to our previous example $\boldsymbol{x} = (0, 0, -11, 10, 7)^t$ with $\boldsymbol{cx} = 0$. We add the two identity moves to get $\boldsymbol{x} + 2\boldsymbol{d}^1 + 2\boldsymbol{d}^2 = (2, 2, 1, 4, 5)^t = C$ and $\boldsymbol{cx} - 2\bar{\boldsymbol{c}}\boldsymbol{d}^1 - 2\bar{\boldsymbol{c}}\boldsymbol{d}^2 = 0 - 4 + 2 = -2$.

Since the $\boldsymbol{d}$'s change basic and non-basic variables to guarantee an integer basis, we need not compute the basic variables using Equation 5.4. We can simply update $\boldsymbol{x_B}$ using the $\boldsymbol{d}$'s. The atomic identity move $\boldsymbol{d}^j$ is column j of the $(n+m)$ by n matrix whose upper part is the n by n identity and whose lower part is $\boldsymbol{V}\hat{\boldsymbol{A}}$ where $\boldsymbol{V}$ is

the diagonal matrix of constraint signs (1 for ≥ and –1 for ≤) and $\hat{\boldsymbol{A}}$ is from Equation 5.1.

Theorem 5.2

The cost of $\boldsymbol{d}^j$ is c_j.

Proof

The cost of $\boldsymbol{d}^j$ is

$$\bar{\boldsymbol{c}}\boldsymbol{d}^j = \bar{c} \cdot \left[\nabla x \middle| \nabla s\right]^t = \bar{c}_j + \sum_{i=1}^{m} \bar{c}_{n+i} \cdot \nabla s_i \qquad (5.13)$$

If constraint *i* is ≥ then $\bar{c}_{n+i} = \pi_i$ and $\nabla s_i = a_{ij}$. If constraint *i* is ≤ then $\bar{c}_{n+i} = -\pi_i$ and $\nabla s_i = -a_{ij}$. So (5.13) becomes

$$\bar{\boldsymbol{c}}\boldsymbol{d}^j = \bar{c}_j + \sum_{i=1}^{m} \pi_i \cdot a_{ij} = \bar{c}_j + \boldsymbol{\pi}\boldsymbol{A}_j = c_j - \boldsymbol{\pi}\boldsymbol{A}_j + \boldsymbol{\pi}\boldsymbol{A}_j = c_j \,\square$$

Theorem 5.2 should not be surprising as $\boldsymbol{d}^j$ only changes one variable with a non-zero cost, x_j. So when we traverse the integer-basis sub-lattice in $\boldsymbol{X_N}$ space via identity moves, we are simultaneously exploring the integer lattice in $\boldsymbol{X}$ space and vice-versa. By viewing the search in terms of $\boldsymbol{X_N}$ space, we gain insight we can use to enhance our search.

Example 5.10

Let $\boldsymbol{G_N}$ = (13, 5) be the incumbent solution. Using Equation 5.4 yields $\boldsymbol{x_B}$ = (3, 1, 2) so $\boldsymbol{G_N}$ is the $\boldsymbol{X_N}$ space image of solution $\boldsymbol{G}$ in $\boldsymbol{X}$ space (Figure 5.2). The cost of solutions $\boldsymbol{G}$ and $\boldsymbol{G_N}$ is -5. If we apply move $\boldsymbol{d}^2 = (0, 1, -3, -2, 1)^t$, we move from $\boldsymbol{G_N}$ = (13, 5) to $\boldsymbol{D_N}$ = (10, 3). Using Equation 5.4 yields $\boldsymbol{x_B}$ = (3, 2, 3) so $\boldsymbol{D_N}$ is the $\boldsymbol{X_N}$

space image of solution $\boldsymbol{D}$ in $\boldsymbol{X}$ space (Figure 5.2). The cost of solutions $\boldsymbol{D}$ and $\boldsymbol{D_N}$ is -4. Move $\boldsymbol{d}^2$ increased x_2 by 1 and did not change x_1 so the change in solution value was $c_2 = 1$ as expected.

Theorem 5.3

Any move from one integer-basis point to another can be expressed as an integer linear combination of the atomic identity moves.

Proof

There is a bijection between the integer points in $\boldsymbol{X}$ space and the integer-basis points in $\boldsymbol{X_N}$ space. Given any integer point in $\boldsymbol{X}$ space, we can calculate the corresponding slack variables. Removing the basic variables gives us the corresponding point in $\boldsymbol{X_N}$ space. Given a point in $\boldsymbol{X_N}$ space, we can calculate the basic variables using Equation 5.4. Extracting the variables in $\boldsymbol{X}$ gives us the corresponding point in $\boldsymbol{X}$ space. Given any two points in $\boldsymbol{X}$ space, the difference between them is an n-dimensional vector that can be expressed as an integer linear combination of n n-dimensional unit vectors. We have n atomic identity moves each changing a single $x_k \in \boldsymbol{X}$ by 1 while fixing all other $x_j \in \boldsymbol{X}$ at their current values so move $\boldsymbol{d}^j$ is equivalent to the n-dimensional unit vector j in $\boldsymbol{X}$ space and the same integer linear combination of atomic identity moves transitions between the images of the two points in $\boldsymbol{X_N}$ space. □

Also by Theorem 5.3, the atomic identity moves provide us with connectivity in our search space.

5.3.3 Identity Move Neighborhoods

The search neighborhood is comprised all of the solutions reachable from the current solution in one iteration. Given a current integer solution $\boldsymbol{x}^r$, a move from the current solution to the next is defined as

$$\boldsymbol{x}^{r+1} = \boldsymbol{x}^r + \sum_{j=1}^{n} p_j \boldsymbol{d}^j \quad p_j \in \quad \forall j = 1...n \tag{5.14}$$

However, evaluating all such combinations is equivalent to enumeration. A larger neighborhood increases the likelihood of finding good solutions but also increases the computational effort required, so we must compromise. We can restrict the neighborhood size by imposing a limit on the atomic identity move coefficients, p_j.

We define a *k-step move* as one where $\sum_{j=1}^{n} |p_j| = k$ then k restricts our neighborhood size. We can have multiple levels of k where k_1+k_2-step implies the restriction $\sum_{j=1}^{n} |p_j| = k_1$ or $\sum_{j=1}^{n} |p_j| = k_2$. We can further restrict the neighborhood size by imposing bounds on the individual coefficients as $l \le p_j \le u$. Of course if $|l| > k$ and/or $|u| > k$ then k becomes the only restriction. Similarly if $k > n|l|$ and $k > n|u|$ then l and u provide the only restrictions. Finally, we can denote an unrestricted parameter as the wildcard *. Therefore a neighborhood N can be defined by (5.14) and the triplet (k, l, u). For example, the neighborhood N(1,-1,1) is comprised of all solutions reached by adding or subtracting a single atomic identity move, the 1-step identity move neighborhood. Regardless of the neighborhood

chosen, the costs of the moves are still fixed. Therefore we can sort them from best to worst and choose the first feasible move.

5.3.4 Bound Strengthening

We can use the reduced costs and the geometry of $\boldsymbol{X_N}$ space to strengthen the bounds on the non-basic variables as the search progresses. Let $\hat{z}_{\text{GMP}}$ be the objective value of the current best solution found to the GMP, $\hat{\boldsymbol{x}}_{\text{GMP}}$. Then $\hat{z}_{\text{GMP}}$ is the *gap* between the objective value of the corresponding integer solution to the ILP and the objective value to the optimal solution of the LP relaxation, z_{LP}^{*}. If $\overline{c}_j > \hat{z}_{\text{GMP}}$ then non-basic variable x_j must be 0 in any solution better than $\hat{\boldsymbol{x}}_{\text{GMP}}$. Otherwise x_j can be nonzero in solutions better than where $\lfloor \hat{z}_{\text{GMP}} / \overline{c}_j \rfloor$ bounds x_j.

5.4 Conclusions

The combinations of non-basic variables that yield an integer basic variables form a sub-lattice in $\boldsymbol{X_N}$ space. The density of this sub-lattice is bounded by the determinant of the LP basis. Any search of the integer feasible sub-lattice in $\boldsymbol{X_N}$ space must incorporate the atomic identity moves. Using the full set of moves provides us with connectivity throughout the solution space (Theorem 5.3). The neighborhoods presented here are general enough to be applied to almost any metaheuristic search method. We present an example Group Theoretic Tabu Search implementation of these ideas in the next chapter.

Chapter 6 - A Group Theoretic Tabu Search Algorithm for the Group Minimization Problem (GTTS-GMP)

In this chapter we develop a group theoretic tabu search (GTTS) algorithm for solving the group minimization problem (GMP), the GTTS-GMP. Given an integer linear program (ILP), we solve a linear programming (LP) relaxation of the ILP and use the LP optimum solution to formulate the GMP. We use identity move neighborhoods to explore the solution space and bound strengthening based on reduced costs as described in Section 5.3.4.

As previously noted, an algorithm that solves the GMP solves the general ILP. Tabu search implementations for the general ILP have been largely ignored. Algorithms developed to exploit the specific characteristics of special case problems tend to be more effective and dominate the literature. However, a general purpose implementation is important to develop techniques that are universal in their effectiveness and to avoid developing a new implementation for each new problem set (Glover and Laguna 1997).

We use multi-dimensional knapsack problems (MDKP), both integer and binary, and set covering problems (SCP), both unicost and weighted, to test the versatility of our approach. GTTS-GMP performs very well on this diverse problem set.

6.1 Methodology

We will first present a short overview of the methodology and provide details on each aspect of the algorithm. At certain points, such as in the discussion of escape procedures, more than one alternative will be described. We will test each of these alternatives and present the results in Section 6.2.

6.1.1 Overview

First, CPLEX solves the linear relaxation of the ILP. The associated LP optimal basic variables, x_{Bi}, and non-basic variables, x_{Nj}, and the reduced costs, $\bar{\boldsymbol{c}}_N$ are used to formulate the GMP. A GTTS-GMP *starting solution,* $\boldsymbol{x}^0$, is generated by rounding the non-slack $\boldsymbol{x}_B$ without regard to constraint feasibility. The start values for the slack variables, both basic and non-basic, are then calculated from $\boldsymbol{Ax}$ and $\boldsymbol{b}$. A 1-step identity move and its inverse is created for each of the original decision variables and placed in move list in ascending order by cost. A reactive tabu search (RTS) procedure is used to explore the solution space using an identity move neighborhood. When cycling is detected an escape procedure is invoked. The algorithm terminates when a GTTS-GMP solution value equal to $\min_j\left(\bar{c}_j\right)$ is found or when the allotted time has expired.

6.1.2 Solving the LP Relaxation

The pseudo code for the initialization phase of the algorithm is shown in Figure 6.1. CPLEX is initialized and the problem data is input. We extract the problem information from CPLEX and relax the integrality restriction on the decision variables. CPLEX is then asked to solve the problem and the solution and reduced cost vectors are retrieved.

In the previous chapter, we assumed the original ILP was a maximization problem with no upper bounded variables. When we remove these assumptions, we introduce some reduced costs that are not positive and some non-basic variables that are not zero (at their lower bound) in the LP optimal solution. If the original problem is a maximization problem, a non-basic variable at its upper bound will have $\bar{c}_j < 0$ while a non-basic variable at its lower bound will have $\bar{c}_j > 0$. If the variable is at its upper bound, $\bar{c}_j$ represents the penalty for *decreasing* the variable by 1. If the variable is at its lower bound, $\bar{c}_j$ represents the penalty for *increasing* the variable by 1. Let ∇x_j be the change in non-basic variable *j*. Our objective function value $z_{\text{GMP}} = \sum_{j=1}^{n} \bar{c}_j \nabla x_j$ regardless of which variables are at their upper or lower bounds. For a minimization problem, the signs on $\bar{c}_j$ are reversed, but z_{GMP} is calculated in the same manner. For maximization $z_{\text{GMP}} \geq 0$ and for minimization $z_{\text{GMP}} \leq 0$. In both cases, $z_{\text{IP}} = z^*_{\text{LP}} - z_{\text{GMP}}$.

The reduced costs returned by CPLEX are prone to round-off and truncation error and occasionally can cause a small valued $\bar{c}_j$ to carry the wrong sign. For this reason, care must be taken to avoid exacerbating this error when using the $\bar{c}_j$. As discussed in Section 5.3.2, to reduce the potential for error, we use the original objective function cost c_j for the move value of each 1-step identity move (Theorem 5.3).

Relax integrality and solve the LP with CPLEX
Retrieve the reduced cost vector, $\bar{\boldsymbol{c}}$, optimal LP solution, $\boldsymbol{x}_{\text{LP}}^{*}$,
    optimal LP objective function value, z_{LP}^{*}, and basis information
Generate the starting solution $\boldsymbol{x}^0$
    Round each non-slack basic variable to the nearest integer
    Calculate the slack variables as $\boldsymbol{s} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$
    Calculate the objective function
        if max $z_{\text{GMP}} = z_{\text{LP}}^{*} - \boldsymbol{c}\boldsymbol{x}^0$
        else $z_{\text{GMP}} = z_{\text{LP}}^{*} + \boldsymbol{c}\boldsymbol{x}^0$
Generate 1-step identity move neighborhood, $N(1, -1, 1)$

Figure 6.1 – Initialization Phase

6.1.3 Finding a Starting Solution

To begin our search, we must have a starting solution. Generating a feasible starting solution for a general IP is not an easy task. Typically, problem specifics must be used to create a feasible starting solution efficiently. We have shown in the previous chapter the use of identity moves allows us to move from a solution outside

the feasible region to a solution inside the feasible region. So, for our search methodology, feasibility of the starting solution is not as important as integrality.

We would also like to be close to $\boldsymbol{x}_{\mathrm{LP}}^{*}$ to hopefully reduce the number of iterations required to move to an area containing good solutions. Perhaps the easiest way to do this is to round $\boldsymbol{x}_{\mathrm{LP}}^{*}$. We round each non-slack basic variable to the *nearest* integer and then recalculate the slack variables as $\boldsymbol{s} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$. This produces an integer solution on a vertex of the hypercube surrounding $\boldsymbol{x}_{\mathrm{LP}}^{*}$ in $\boldsymbol{X}$ space. The solution may or may not be feasible.

6.1.4 Identity Move neighborhoods

Theorem 5.3 assures that any neighborhood containing the 1-step identity move neighborhood, $N(1,-1,1)$, will guarantee connectivity throughout the solution space. Futher, Theorem 5.3 assures any identity move neighborhood is a composite of $N(1,-1,1)$. However, larger composite move neighborhoods *may* perform better because they reach more solutions in a single iteration. The size of $N(1,-1,1)$ is simply $2n$.

The full 2-step identity move neighborhood, $N(2,-2,2)$, contains all solutions reached from the current by combining any 2 1-step moves. The size of $N(2,-2,2)$ is $\binom{2n}{2} - n + 2n = \binom{2n}{2} + n$. Since a small problem with n = 1000 implies a neighborhood with precisely 2 million members, using $N(2,-2,2)$ is not practical for most problems.

Candidate lists are often used in tabu search to restrict neighborhood size (Glover and Laguna 1997). For example, in a TSP we may restrict swap moves to only consider swapping cities within ten positions of each other in the current solution. Such a restriction is often arbitrary, but is valid as long as connectivity within the solution space is maintained. Using $N(1,-1,1)$ in conjunction with a candidate list comprised of any subset of $N(2,-2,2)$ assures connectivity.

One possible candidate list is constructed as follows: create $N(1,-1,1)$ and sort the moves in ascending order by move value. (This list is symmetric: the bottom of the list is the inverse of the top.) Next combine each move with the 2 moves immediately preceding and following its inverse in the list. This strategy excludes the extreme combinations, combining two really good moves or two really bad moves, and creates "fine-tuning" moves with smaller move values. The number of combined moves is $2n - 2$, so the size of the neighborhood is $2n + 2n - 2 = 4n - 2$.

Example 6.1

$N(1,-1,1)$

Move 3	-150
Move 5	-140
Move 2	-115
Move 1	-90
Move 4	-80
Move -4	80
Move -1	90
Move -2	115
Move -5	140
Move -3	150

Subset of $N(2,-2,2)$

Move 5, -2	-25
Move 2, -1	-25
Move 3, -5	-10
Move 1, -4	-10
Move 4, -1	10
Move 5, -3	10
Move 1, -2	25
Move 2, -5	25

An alternate approach is to create combination moves dynamically during the search. Given that the best feasible non-tabu move is taken at each iteration and that the values of the moves are fixed, if the move just executed is feasible it will be chosen again. This will continue until the move is no longer feasible. In other words, we have reached a boundary of the feasible region. So changes in a chosen move occur at the boundaries of the feasible region.

We can create $N(1,-1,1)$ as described above and begin our search. When the current 1-step move being executed is different than the previous one, we combine the two moves and add the new composite move (and its inverse) to the move list. Our algorithm now learns the "shape" of the solution space. Of course, a restriction will need to be in place to limit the number of dynamic moves created in this manner.

Regardless of the neighborhood used, the moves are sorted in ascending order and the first (best) feasible non-tabu move is chosen. If a feasible non-tabu move cannot be found, the first infeasible non-tabu move is chosen. If the current solution is infeasible, the first non-tabu move reducing infeasibility is chosen. If a non-tabu move reducing feasibility cannot be found, the first non-tabu move is chosen.

Finally, if two or move moves have the same value and are non-tabu the move with the greatest normalized-surplus is chosen. The normalized surplus is sum of the surplus in the constraints, each normalized by their respective right hand side values. We are essentially choosing the more "interior" solution. Any further ties are broken lexicographically. In our testing, we compare the dynamic approach versus the basic $N(1,-1,1)$ neighborhood.

6.1.5 Tabu Search and Tabu Structures

The pseudo code for the search phase is shown in Figure 6.2. When using a search neighborhood, we may become trapped in local optima. Tabu search (TS) helps us to escape local optima by allowing non-improving moves and making recently visited solutions tabu (Glover and Laguna 1997).

After a move is executed an attribute of the current solution or move is added to the *tabu list*. For a number of iterations, *tabu tenure*, solutions or moves on that list are not permitted unless such a move leads to a solution superior to any found to that point in the search. Solution attribute strategies typically allow a more flexible search than move based strategies.

From Section 5.3.3, a move neighborhood is specified by the triplet (k, l, u) where k is the sum of the multipliers for the atomic moves, l is the lower bound of the multipliers and u is the upper bound of the multipliers. If $l = -u$ then we have a symmetric neighborhood. When a move is performed we can add its inverse move to the list. This tabu strategy would work for simple neighborhoods like $N(1, -1, 1)$, but for more complex neighborhoods where $k > 1$ there will be more than one way to return back to the same solution. For such neighborhoods a solution attribute is *required* if returning to a previously visited solution within its tabu tenure is to be prohibited.

In a reactive tabu search (RTS) we also implement a tabu structure to track the frequency in which solutions are visited (Battiti and Tecchiolli 1994). This structure

is used to detect the whether the search is trapped in an attractor basin and to adjust the tabu tenure or implement escape procedures when such a basin is detected. When a solution is repeated within a predetermined interval the tenure is increased to facilitate diversification into new areas of the solution space. When a predetermined number of consecutive new solutions are encountered, tenure is reduced to facilitate intensification of the search within the current subset of the solution space. When a predetermined number of solutions have been repeated, we conclude the presence of an attractor basin and an escape procedure is invoked.

Due to the immense size of the solution space, we can not explicitly store each solution encountered. Further, since all previous solutions must be checked at every iteration we must have an efficient way to store and access solutions. Just as in Section 4.2.5, a hash function (Woodruff and Zemel 1993), $\varphi(\boldsymbol{x})$, is used. A random number, ρ_j, is generated for each x_j. The solution's hash value is $\varphi(\boldsymbol{x}) = \sum_{j=1}^{n} \rho_j x_j$. Solution information, such as number of visits and iteration last visited, is then stored in a table based on $\varphi(\boldsymbol{x})$. When two or more distinct solutions produce the same hash value, a *collision* occurs. We can limit collisions by further distinguishing a solution by its deficit or surplus.

We can implement a solution-based tabu strategy using the solution tabu structure and $\varphi(\boldsymbol{x})$. Since $\varphi(\boldsymbol{x})$ is an additive function, we can calculate the change in $\varphi(\boldsymbol{x})$, $\Delta\varphi(\boldsymbol{x})$, for each move. As we evaluate each move, we can use $\Delta\varphi(\boldsymbol{x})$,

the solution tabu structure and the current solutions hash value to determine if the move leads to a recently visited solution.

We use a combination of move-based and solution-based tabu strategies in our algorithm. A move is tabu if the move's inverse has been executed in MOVE_TENURE iterations or if the solution has been visited in SOL_TENURE iterations. If a move leads to a new best found solution, tabu status is ignored.

```
Search loop iteration i
        If x^i is feasible
                Find the first feasible non-tabu move
        else
                Find the first non-tabu move the reduces infeasibility
        Execute the move
        If x^(i+1) feasible and better than best found
                Update best found
                Check for optimality
                Update bounds
        If using dynamic neighborhoods
                If the current move is different then the previous one
                        Combine the moves and add the new move to the list
        Update move and solution tabu structures and get search status
        If status is intensification
                Decrease tabu tenure
        If status is diversification
                Increase tabu tenure
        If status is cycling
                Escape
        If stopping criteria not met repeat search loop
Return best solution found
```

Figure 6.2 – Search Phase

6.1.6 Escape Procedures

We considered two escape strategies – random restart and using a surrogate objective function. Both escape strategies strive to break the current cycle or capture basin while returning the search to the vicinity of $\boldsymbol{x}_{\mathrm{LP}}^{*}$.

For random restart, we round $\boldsymbol{x}_{\mathrm{LP}}^{*}$ in a random fashion. For each non-integer x_j the probability it is rounded to $\lceil x_j \rceil$ is $\lceil x_j \rceil - x_j$ and the probability it is rounded to $\lfloor x_j \rfloor$ is $x_j - \lfloor x_j \rfloor$. The current solution is then set to the newly rounded $\boldsymbol{x}_{\mathrm{LP}}^{*}$ solution. Again this solution may or may not be feasible. The move-based tabu information is reset as it was based on the previous location and the search is resumed. When generating a random solution, we do not allow a restart solution to be a solution that has been visited more than MAX_RESTART_REPEAT times.

For the surrogate objective function strategy, we change our objective to that of minimizing the Euclidean distance to $\boldsymbol{x}_{\mathrm{LP}}^{*}$. The surrogate objective function is

$$\min \quad \sum_{j=1}^{n} \left(x_j - x_{LP\,j}^{*} \right)^2 \tag{6.1}$$

Minimizing Equation 6.1 requires us to search the entire neighborhood not just the top of the sorted list. We do require the x_j to be within their bounds but do not require the solution to be feasible with respect to the constraints.

Of course, an optimal solution in terms of Equation 6.1 is our starting solution, $\boldsymbol{x}^0$. However, the tabu structure keeps the escape procedure from converging to the same solution each time and we do not continue the escape

procedure for very long. We minimize with respect to Equation 6.1 until a local optima is reached or MAX_ESCAPE_ITER iterations have been performed.

6.1.7 Cutting Planes

Since we are using $\boldsymbol{x}_{\mathrm{LP}}^{*}$ to guide the search, it makes sense to consider adding cutting planes to the LP to move $\boldsymbol{x}_{\mathrm{LP}}^{*}$ closer to the optimal IP solution and hopefully improve our guide. There are many types of cutting planes each with their own advantages and disadvantages depending on the problem being solved. Instead of developing an algorithm to generate and implement of these cuts, a massive undertaking, we simply let CPLEX apply the cuts.

CPLEX provides several *callback* routines that allow an algorithm to interact with CPLEX's solvers as they are executing. The user provides CPLEX the address of a subroutine to be called when a specific event occurs. When the user's routine is completed, control is passed back to CPLEX (unless the user terminates it).

One such callback for the mixed integer program optimizer (mipopt) is the *heuristic callback* which calls the user's specified routine at every viable node (feasible, not fathomed) in the branch and bound tree. The solution to the current LP relaxation, $\boldsymbol{x}_{\mathrm{LP}}^{*}$, as well as pointers that can be used to access the other information from the current LP is passed to the user's routine. At node 0 in mipopt, CPLEX adds cuts and solves the node LP repeatedly until the cuts no longer make significant progress. CPLEX invokes the heuristic callback after solving each node LP.

We call CPLEX's mipopt to solve our problem. Cuts added at node 0 are global cuts applying to the whole problem. We retrieve the new solution, basis status, and reduced costs after every node 0 LP. Once node 0 is completed, we use the final node LP as our $\boldsymbol{x}_{\mathrm{LP}}^{*}$ and terminate CPLEX. Which $\boldsymbol{x}_{\mathrm{LP}}^{*}$ we used affects the starting point, the escape procedure, and bound strengthening. We test our algorithm using the $\boldsymbol{x}_{\mathrm{LP}}^{*}$ *with and without cuts* to determine if the cuts do indeed improve performance.

6.1.8 Bound Strengthening

We employ the bound strengthening in our algorithm as described in Section 5.3.4. Given an objective function value z_{GMP}, the bounds on x_j are $lb_j = \min\left(lb_j, ub - \left(\left\lfloor z_{GMP}/\overline{c}_j \right\rfloor + 1\right)\right)$ for upper bound non-basic variable j and $ub_j = \min\left(ub_j, lb + \left\lfloor z_{GMP}/\overline{c}_j \right\rfloor\right)$ for lower bound non-basic variable j. The bounds are updated whenever a new best solution is discovered.

CPLEX also performs bound strengthening as part of the pre-processing for each node. We can use a callback function as described in the previous section to retrieve the strengthened bounds from node 0 in the mipopt algorithm. The details of the bound strengthening performed by CPLEX are not provided. The reduced cost based bound strengthening is implemented in all versions of GTTS-GMP tested. The algorithms are tested with and without the CPLEX bounds to determine their effect on performance.

6.1.9 Strategic Oscillation

Strategic oscillation is allowing the search to exit the feasible region at strategic points then driving it back inside in hopefully a new area of the feasible solution space. We maintain bounds feasibility of the decision variables throughout the algorithm. However, our initial solution may be infeasible in terms of the constraint set. We attempt to reduce infeasibility at each iteration until the feasible region is reached. Once feasible, we allow only feasible moves. If the neighborhood does not contain a feasible non-tabu move the best infeasible non-tabu move is chosen.

We incorporate strategic oscillation in both escape procedures. In the random restart strategy, the random starting solution is most likely infeasible. We then move towards the feasible region as described above. In the surrogate objective function strategy, we completely ignore feasibility of the constraint set as we move towards $\boldsymbol{x}_{\mathrm{LP}}^{*}$ allowing the search to exit the feasible region if it improves the objective.

6.1.10 Stopping Criteria

Since we are using a search heuristic, it is unlikely we will be able to prove that a global optimal solution has been found. Since some of our basic variables are fractional we know at least one of the non-basic variables must change by at least 1. Therefore, $z_{\mathrm{LP}}^{*} - \min_{j}(\bar{c}_j)$ is an upper bound on a global optimal solution for the ILP.

If we find a feasible solution that equals this bound then we can terminate with a global optimal solution. Otherwise, we continue to search until a maximum time limit or number of iterations has been reached.

6.2 Computational Results

6.2.1 Test Cases

We use set covering and multi-dimensional knapsack problem sets to test our algorithm. These problems contain a diverse set of characteristics. SCPs are minimization problems and MDKPs are maximization problems. SCPs are binary while MDKPs can be binary or general integer programming problems. We solve both versions here. SCPs have relatively sparse constraint matrices consisting of all 1s and 0s while MDKPs have relatively dense matrices containing almost any values.

problem set	number of problems	number of columns	number of rows	density
4	10	1000	200	2%
5	10	2000	200	2%
6	5	1000	200	5%
A	5	3000	300	2%
B	5	3000	300	5%
C	5	4000	400	2%
D	5	4000	400	5%
E	5	500	50	20%
NRE	5	5000	500	10%
NRF	5	5000	500	20%
NRG	5	10000	1000	2%
NRH	5	10000	1000	5%

Table 6.1 Characteristics for SCP sets

The benchmark problems solved were obtained from Beasley's OR-Library (Beasley 1990). For set covering, problem sets 4-6 originally appeared in (Balas and Ho 1980), problem sets A-E appeared in (Beasley 1987) and problem sets NRE-NRH appeared in (Beasley 1990). All problems were randomly generated based on the strategy of (Balas and Ho 1980). All of the problem sets except E were generated as weighted SCPs. Problem set E is a unicost SCP. The characteristics of the SCP sets are shown in Table 6.1.

For knapsack, problem set 1 appeared in (Petersen 1967), problem sets CB1-CB9 appeared in (Chu and Beasley 1998). All problems were generated as binary problems; however, we solve each problem as binary and as general integer. Problem set 1 contains small easy to solve problems. Problem sets CB1-CB9 get progressively more difficult. There are 30 problems in each set. We solve the first problem in sets CB1-CB8 and the first 10 problems in CB9. The characteristics for the MDKP sets are shown in Table 6.2.

problem set	number of problems	number of columns	number of rows	binary
1	7	6-50	5-10	n
1b	7	6-50	5-10	y
cb11-cb81	8	100-500	5-30	n
cb11b-cb81b	8	100-500	5-30	y
cb91-cb910	10	500	30	n
cb91b-cb910b	10	500	30	y

Table 6.2 Characteristics for MDKP sets

6.2.2 Test Procedures

All tests were performed on Dell Precision 530 Workstations running SuSE Linux with two 1.8GHz Pentium Xeon processors and 1GB of RAM. The machines

are multi-user platforms. An attempt was made to find machines that were not too busy, but as some problems ran for at least an hour, CPU usage surely fluctuated during processing. Each problem was solved using CPLEX 9.0 and GTTS-GMP with a time limit of 3600 seconds for either method. GTTS-GMP was coded in C.

6.2.3 CPLEX 9.0

To provide a high benchmark, we compare our results to CPLEX version 9.0. CPLEX uses a very sophisticated branch and cut algorithm to solve MILPs. The algorithm is further aided by two heuristics. The first attempts to create a feasible solution from the fractional solution at the node. The second attempts to improve the incumbent integer solution through a neighborhood search (ILOG 2003).

It would be naïve to think that we could outperform CPLEX with a general ILP algorithm. CPLEX was developed with millions of dollars and man-centuries of effort. CPLEX achieves the optimal or best known solutions for nearly every test problem. We simply hope to perform well against this benchmark to demonstrate the potential of our approach.

6.2.4 Results

In our testing, we found that the dynamic neighborhoods performed better than the 1-step neighborhoods on the MDKP problem sets. However, we found the opposite to be true for the SCP problems sets. This may be a factor of neighborhood size. The largest MDKP test instance has 500 columns while the smallest SCP test

instance has 1000 columns. So for the SCP sets the 1-step neighborhoods are significantly larger than for the MDKP sets.

We used the dynamic neighborhood option for the MDKP sets and the 1-step neighborhood for the SCP sets for all algorithm configurations. The tabu tenure also differs between problem sets. The default move tenure is a function of problem size, n*0.1. The default solution tabu tenure is smaller when dynamic neighborhoods are used. It makes sense to revisit solutions more frequently when the neighborhood has changed. Since we are using reactive tabu search, both tabu tenures change throughout the search.

We next examine the surrogate objective function escape strategy. The results for the SCP sets are presented in Table 6.3. The results for the MDKP problem sets are presented in Table 6.4. We tested each problem with cuts and bounds from CPLEX, with cuts only, with bounds only, and without cuts or bounds.

With the exception of the smaller problems adding cuts and/or bounds from CPLEX improves the performance on the SCP sets. The bounds from CPLEX for SCPs are generally stronger than those derived using reduced costs. Also the algorithm has these bounds at the start and does not need to wait for new solutions to generate them. The bounds seem to affect performance more than the cuts. All things considered, GTTS-GMP with cuts and bounds performs best.

For the MDKP sets the effect of bounds and cuts is less significant. For the general integer problems, the bounds from reduced cuts quickly dominate those from CPLEX. In either case the bounds from CPLEX are not as strong as they are for the

SCP sets. In general, the cuts do not have as much of an impact on the algorithm as we anticipated. However, we still feel that GTTS-GMP with cuts and bounds performs best.

problem set	optimal or *best known average	CPLEX average (time sec)	GTTS-GMP average - (time in seconds) - % gap			
			cuts & bounds	cuts only	bounds only	none
4	510	510 (0)	511.2 (10.2) 0.22%	512.3 (3.7) 0.45%	511.3 (0.5) 0.25%	510.4 (0.9) 0.08%
5	257.2	257.2 (0.3)	259.2 (8) 0.78%	259 (12) 0.70%	259.2 (6.9) 0.78%	258.8 (6.5) 0.62%
6	144.2	144.2 (0)	147.8 (41.6) 2.5%	148.2 (32.2) 2.77%	146.6 (6.8) 1.66%	148.2 (5.4) 2.77%
A	241.4	241.4 (1.2)	245.6 (54.2) 1.74%	246.8 (69) 2.24%	246.4 (73.2) 2.07%	246.6 (19.2) 2.15%
B	75.2	75.2 (1.8)	76.8 (56.8) 2.13%	77.8 (44.8) 3.46%	77 (61.2) 2.39%	77.4 (29.4) 2.93%
C	224.6	224.6 (1.8)	233 (88) 3.74%	233.4 (107.2) 3.92%	232.4 (41.4) 3.47%	232.6 (131) 3.56%
D	64.2	64.2 (8)	66.8 (65.8) 4.05%	66.4 (117.2) 3.43%	67.4 (55) 4.98%	67 (138) 4.36%
E	5	5 (0)	5 (0) 0.0%	5 (0) 0.0%	5 (0) 0.0%	5 (0) 0.0%
NRE	28.4	28.4 (390.2)	29.4 (147) 3.52%	29.6 (291.4) 4.23%	29 (785.8) 2.11%	29.8 (56.4) 4.93%
NRF	14	14 (721)	14.2 (14) 1.43%	14.8 (105.2) 5.71%	14 (621.2) 0.0%	15 (8.8) 7.14%
NRG	*166.4	166.8 (625.6)	180 (479) 8.17%	181.6 (1180) 9.13%	180 (430.6) 8.17%	180.6 (1522) 8.53%
NRH	*59.6	61.2 (1361.2)	65.8 (183) 10.40%	66 (798.6) 10.74%	66 (226.8) 10.74%	68 (208) 14.09%

Table 6.3 SCP results for surrogate objective function escape strategy

problem set	optimal or *best known average	CPLEX average (time sec)	GTTS-GMP average (time in seconds)			
			cuts & bounds	cuts only	bounds only	none
1	13604.41	13604.41 (0)	13604.27 (0) 0.0%	13603.56 (0.3) 0.01%	13603.56 (0.6) 0.01%	13604.27 (0.1) 0.0%
1b	8885.16	8885.16 (0)	8883.16 (0.1) 0.02%	8883.16 (1.3) 0.02%	8885.16 (0) 0.0%	8885.16 (1.4) 0.0%
cb11-cb18	*78337.5	78337.5 (166.5)	78173.63 (119.5) 0.21%	78157.63 (311.3) 0.23%	78179.75 (430.3) 0.20%	78179.63 (295) 0.20%
cb11b-cb18b	*60330.13	60330.13 (441.38)	60192.5 (515.9) 0.23%	60188 (686.7) 0.24%	60202 (849.5) 0.21%	60183.13 (510.6) 0.24%
cb91-cb910	*125682.8	125682.8 (1773.5)	125194.6 (1118.6) 0.39%	125165.4 (1391.5) 0.41%	125163.2 (1020.9) 0.41%	125154.1 (1211) 0.42%
cb91b-cb910b	*115553.9	115553.9 (1736.9)	115239 (1357) 0.27%	115229.4 (1348.6) 0.28%	115227.5 (918.2) 0.28%	115232.8 (1122.5) 0.28%

Table 6.4 MDKP results for surrogate objective function escape strategy

Next we examine the random restart escape strategy. The results for the SCP sets are presented in Tables 6.5. The results for the MDKP problem sets are presented in Tables 6.6. We tested each problem with cuts and bounds from CPLEX, with cuts only, with bounds only, and without cuts or bounds. The results are the average of three runs for each configuration.

With the exception of set 1 adding cuts and/or bounds from CPLEX improves the performance on the SCP sets. The bounds have a much greater impact on performance than the cuts. One possible reason for this is that the cuts reduce the fractional parts of $\boldsymbol{x}_{\mathrm{LP}}^{*}$ which then limits the variability of the random restart solutions. On average we still feel that GTTS-GMP with cuts and bounds performs best.

Again for the MDKP sets the effect of bounds and cuts is less significant for the same reasons as above. GTTS-GMP with cuts and bounds also performs best on he MDKP sets.

problem set	optimal or *best known average	CPLEX average (time sec)	GTTS-GMP average - (time in seconds) - % gap			
			cuts & bounds	cuts only	bounds only	none
4	510	510 (0)	512.37 (6.3) 0.46%	511.9 (10.1) 0.37%	514.06 (3.23) 0.80%	510.2 (12.5) 0.04%
5	257.2	257.2 (0.3)	258 (17.5) 0.31%	258.33 (18) 0.44%	258.6 (24.4) 0.54%	258.17 (15.9) 0.38%
6	144.2	144.2 (0)	145.13 (19.4) 0.65%	145.6 (41.9) 0.97%	145 (31.5) 0.55%	145.53 (36.7) 0.92%
A	241.4	241.4 (1.2)	243.6 (79.9) 0.91%	243.93 (103.3) 1.05%	244.2 (84.1) 1.16%	244.8 (101.4) 1.41%
B	75.2	75.2 (1.8)	75.87 (57.9) 0.89%	76.2 (60.1) 1.33%	75.8 (105.2) 0.80%	76.27 (85.5) 1.42%
C	224.6	224.6 (1.8)	230.53 (108) 2.64%	230.53 (133.5) 2.64%	230.33 (78.9) 2.55%	231.33 (124.1) 3.00%
D	64.2	64.2 (8)	65.13 (92.3) 1.45%	65.73 (111.7) 2.39%	65.47 (88.5) 1.97%	65.87 (80.1) 2.60%
E	5	5 (0)	5 (0) 0.0%	5 (0) 0.0%	5 (0) 0.0%	5 (0) 0.0%
NRE	28.4	28.4 (390.2)	28.6 (433.3) 0.70%	29 (146.67) 2.11%	28.53 (930) 0.47%	28.73 (654.5) 1.17%
NRF	14	14 (721)	14.07 (341.1) 0.48%	14.2 (62.8) 1.43%	14.13 (239.1) 0.95%	14.07 (345.1) 0.48%
NRG	*166.4	166.8 (625.6)	179.07 (605.3) 7.61%	178.93 (612.1) 7.53%	178.5 (1132.3) 7.29%	178.4 (391.3) 7.21%
NRH	*59.6	61.2 (1361.2)	63.67 (1078.1) 6.82%	63.33 (1015.8) 6.26%	63.73 (564.3) 6.94%	63.73 (1241.9) 6.94%

Table 6.5 SCP results for random restart escape strategy

problem set	optimal or *best known average	CPLEX average (time sec)	GTTS-GMP average (time in seconds)			
			cuts & bounds	cuts only	bounds only	none
1	13604.41	13604.41 (0)	13604.03 (0.1) 0.0%	13604.03 (0.3) 0.0%	13603.27 (0.4) 0.00%	13600.6 (0.8) 0.03%
1b	8885.16	8885.16 (0)	8884.49 (0.2) 0.01%	8883.16 (0.1) 0.02%	8883.59 (1.1) 0.02%	8877.97 (0.2) 0.08%
cb11-cb18	*78337.5	78337.5 (166.5)	78199.42 (222.6) 0.18%	78205.83 (307.8) 0.17%	78198.92 (341.1) 0.18%	78200.96 (540.9) 0.17%
cb11b-cb18b	*60330.13	60330.13 (441.38)	60225.5 (555.6) 0.17%	60229.46 (513.6) 0.17%	60205.5 (452.9) 0.21%	60198.21 (669.83) 0.22%
cb91-cb910	*125682.8	125682.8 (1773.5)	125379.7 (1685.8) 0.24%	125382.7 (1374.7) 0.24%	125398.3 (1384.9) 0.23%	125372.8 (1782.5) 0.25%
cb91b-cb910b	*115553.9	115553.9 (1736.9)	115361.9 (1615.9) 0.17%	115323.8 (1311.4) 0.20%	115336.6 (1511.5) 0.19%	115313.1 (1781.1) 0.21%

Table 6.6 MDKP results for random restart escape strategy

The random restart escape strategy is clearly superior to the surrogate objective function. Although both strategies are able to escape from cycling, the surrogate objective function drives the search towards the same point, $\boldsymbol{x}^0$, every time. While the tabu structure stops the procedure from visiting the same solution from one escape to the next, it still does not provide the diversification that we get from the random restart strategy.

6.3 Super Optimal Solutions

Our algorithm oscillates in and out of the feasible region during the search. Often high quality solutions can be found with very minor violations in feasibility. We call such solutions *super-optimal* (Carlton and Barnes 1996). The constraints in

our models are often the result of some simplifying assumptions and are rarely pristine. The decision maker may want to evaluate these super-optimal solutions to determine if they could accept the violations given the value of the solution. Tables 6.7 and 6.8 contain examples of the super-optimal solutions found. For MDKP the violation is normalized by the right hand side. For example, if the constraint is ≤ 500 and the value is 525 then the violation is 25/500 or 0.05.

problem	optimal	super-optimal	violation
44	497	470	1
49	641	630	1
58	288	265	1
64	131	123	1

Table 6.7 Some super-optimal solutions from SCP sets

problem	optimal	super-optimal	violation
12	10970.9	11082.7	0.004
12b	8706.1	8886.2	0.009
cb21	93127	93216	0.0005
cb31	175856	176082	0.0009
cb71b	21946	22008	0.007

Table 6.8 Some super-optimal solutions from MDKP sets

6.4 Conclusion

Our GTTS-GMP algorithm performs well, finding solutions well within 5% of the best known for all but 2 problem sets. Many of these techniques, rounding the LP relaxation, bounding by reduced costs, using cuts, etc., are universal and can be applied to tailored algorithms designed for specific problems.

For our test, the random restart strategy with cuts and variable bounds from CPLEX performed the best overall. More research needs to be done to improve the performance of the general algorithm. However, even if a general algorithm is used it

is clear that tuning parameters, such as dynamic neighborhood and tenure, for a specific problem type will improve performance.

Chapter 7 - Conclusion & Future Research

7.1 Conclusions

In this work we explored the use of group theory in metaheuristic search methods for partitioning integer linear programs (ILPs). To the best of our knowledge, this is the first time local search methods for these types of problems have been viewed through this framework. Past efforts have demonstrated the power of group theory in metaheuristic search methods for problems with an ordering component.

7.1.1 Partitioning into Orbits

Using group theory we defined procedures for partitioning the solution space into orbits. By clustering the variables, we are able to create "good" and "bad" orbits. We developed neighborhoods to explore the individual orbits and transition between them. We also developed methods for bounding the solutions in each orbit so that the may be discarded as infeasible or dominated by the incumbent solution.

We tested these techniques by developing a group theoretic tabu search algorithm to solve the unicost set covering problem, the GTTS-USCP. Our variable clustering was based on the reduced costs of the LP relaxation of the problem. The use of variable clustering and group theory allowed our algorithm to intensify the search in the areas of the solution space believed to contain good solutions. The orbits kept the search contained in these areas and the clusters worked as an enhanced

candidate list, reducing the total number of moves in the neighborhood while still retaining the "good" moves. These techniques proved very effective for the unicost SCP discovering 46 new best known solutions for the set of 65 benchmark problems and outperformed CPLEX in both solution quality and speed.

7.1.2 The Group Minimization Problem

Next we looked at the general ILP by examining its associated group minimization problem (GMP). We demonstrated why valid column reduction for the GMP is not valid in terms of the original ILP. We examined the corner polyhedron in the space of the set of non-basic variables. We proved that the density of the integer points in non-basic variable space that yield an all-integer basis is bounded by the determinant of the basis. We defined new search neighborhoods, identity move neighborhoods, to traverse the sub-lattice formed by these points. Finally, we developed procedures for strengthening the bounds on the non-basic variables using the reduced cost from the optimal solution to the LP relaxation. These bounds reduce the size of the corner polyhedron and the solution space of the GMP.

Based on these results we developed a GTTS algorithm for the GMP, GTTS-GMP. Since a GMP can be formed for any ILP, our algorithm solves the general ILP. We are able to add cuts and additional variable bounds from CPLEX at the beginning of our algorithm. The algorithm performs well against a diverse set of test problems.

7.2 Future Research

7.2.1 Other Clustering Techniques

For the unicost SCP problem, the LP solution provided a good profile of quality ILP solutions and provided a good basis for variable clustering. This is not likely to be true for all problems types. In addition, for problems where solving the LP relaxation is not reasonable other methods of clustering will need to be developed. Heuristic methods, such as benefit/cost ratio for knapsack problems, may provide a more efficient and accurate clustering and should be explored.

Another possible future enhancement to the partitioning algorithm is the use of composite moves. By using only single transpositions, we only implemented simple swap moves in our move neighborhoods. It might be worthwhile to create composite moves by combining transpositions from each cluster.

7.2.2 Embed GTTS-GMP in Branch & Cut

GTTS-GMP could be embedded within a branch & cut program such as CPLEX. The branch & cut algorithm could execute a short version of the GTTS-GMP algorithm at each viable node to attempt to find a new feasible incumbent solution. The local bounds at the node provide diversification for the GTTS-GMP algorithm and the solutions found by GTTS-GMP help the branch & cut algorithm to fathom more nodes. We feel this concept holds a lot of promise and should be explored further.

7.2.3 Equality Constraints

The method for creating identity moves presented in Section 5.3.2 does not consider equality constraints. One approach may be to replace each equality constraint with a $\geq$ and $\leq$ constraint and proceed as normal. This may present difficultly in maintaining a feasible solution during the search. Another approach may be to create the 1-step neighborhood ignoring the equality constraints and penalize the search based on the violation of these constraints. It may also be possible to use the coefficients in the equality constraints to combine 1-step moves into composite moves that maintain the feasibility of these constraints. An efficient and effective method for dealing with these constraints is another area of research.

7.2.4 Mixed Integer Linear Programs

The GTTS-GMP is for all integer problems. How can we apply these techniques to mixed integer problems? There is a mixed integer version of the GMP developed by Araoz (1973). The columns of his problem form an abelian *semigroup*. A semigroup lacks the inverse and identity properties of a group. Perhaps the semigroup minimization problem can help us develop techniques for the mixed integer case.

The research documented here was supported by a grant for the Air Force Office of Scientific research.

Bibliography

Aráoz, J. 1973. *Polyhedral Neopolarities.* PhD Dissertation, Dept of Computer Science and Applied Analysis. University of Waterloo. Waterloo, Ontario.

Aráoz, J., L. Evans, R.E. Gomory, and E.L. Johnson. 2003. Cyclic group and knapsack facets. *Mathematical Programming* 96(2): 377-408.

Balas, E. 1973. A Note on the Group Theoretic Approach to Integer Programming and the 0-1 Case. *Operations Research* 21(1): 321-322.

Balas, E., A. Ho. 1980. Set Covering Algorithms Using Cutting Planes, Heuristics, and Subgradient Optimization: a Computational Study. *Mathematical Programming* 12: 37-60.

Balas, E., M.C. Carrera. 1996. A Dynamic Subgradient-Based Branch and Bound Procedure for Set Covering. *Operations Research* 44: 875-890.

Barnes, J.W., B.W. Colletti, and D.L. Neuway. 2002. Using Group Theory and Transition Matrices to Study a Class of Metaheuristic Neighborhoods. *European Journal of Operational Research* 138: 531-544.

Barnes, J.W., B. Dimova, S.P. Dokov and A. Solomon. 2003. The Theory of Elementary Landscapes. *Applied Mathematical Letters* 16: 337-343.

Barnes, J.W., V.D. Wiley, J.T. Moore and D.M. Ryer. 2004. Solving the Aerial Fleet Refueling Problem using Group Theoretic Tabu Search. *Mathematical and Computer Modeling* 39: 617-640.

Battiti, Roberto and G. Tecchiolli. 1994. The Reactive Tabu Search. *ORSA Journal on Computing* 6(2): 126-140.

Beasley, J.E. 1987. An Algorithm for Set Covering Problem. *European Journal of Operational Research* 31: 85-93.

Beasley, J.E. 1990. A Lagrangian Heuristic for Set Covering Problems. *Naval Research Logistics* 37: 151-164.

Beasley, J.E. 1990. OR-library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society* 41(11): 1069-1072.

Beasley, J.E., A. Chu. 1996. Genetic Algorithm for the Set Covering Problem. *European Journal of Operational Research* 94: 392-404.

Carlton, William B. and J.W. Barnes. 1996. Solving the traveling-salesman problem with time windows using tabu search. *IIE Transactions* 28(8): 617-630.

Ceria, S., P. Nobili, A. Sassano. 1998. A Lagrangian-Based Heuristic for Large-Scale Set Covering Problems. *Mathematical Programming* 81: 215-228.

Chu, P.C. and J.E.Beasley. 1998. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics* 4: 63-86.

Chvátal, V. 1979. A Greedy Heuristic for the Set Covering Problem. *Mathematics of Operations Research* 4(3): 233-235.

Codenotti, Bruno and L. Margara. 1992. Local Properties of Some NP-Complete Problems. *TR-92-021 (March 1992)*, International Computer Science Institute, University of California at Berkeley.

Colletti, Bruce W. 1999. *Group Theory and Metaheuristics.* Ph.D. Dissertation, The University of Texas at Austin.

Colletti, Bruce W. and J. W. Barnes. 2000. Linearity in the Traveling Salesman Problem. *Applied Mathematical Letters* 13: 27-32.

Colletti, Bruce W. and J.W. Barnes. 2001. Local search structure in the symmetric traveling salesperson problem under a general class of rearrangement neighborhoods. *Applied Mathematical Letters* 14: 105–108.

Combs, T.E. 2002. *A Combined Adaptive Tabu Search and Set Partitioning Approach for the Crew Scheduling Problem with an Air Tanker Crew Application*. Ph.D. Dissertation, Air Force Institute of Technology.

Combs, T.E., J.T. Moore. 2004. A Hybrid Tabu Search/Set Partitioning Approach to Tanker Crew Scheduling, *Military Operations Research Society Journal* 9: 43-57.

Crino, John R. 2002. *A Group Theoretic Tabu Search Methodology for Solving Theater Distribution Vehicle Routing and Scheduling Problems.* Ph.D. Dissertation, Air Force Institute of Technology.

Crino, J.R., J.T. Moore, J.W. Barnes and W.P. Nanry. 2004. Solving the Theater Distribution and Scheduling Problem using Group Theoretic Tabu Search. *Mathematical and Computer Modeling* 39: 599-616.

Daskin, M.S., E. Stern. 1981. A Hierarchical Objective Set Covering Model for EMS Vehicle Deployment. *Transportation Science* 15: 137-152.

Dimova, Boryana, J.W. Barnes and E Popova. 2005. Arbitrary Elementary Landscapes and AR(1) Processes. *Applied Mathematical Letters* 18: 287-292.

Fraleigh, John B. 1976. *A First Course in Abstract Algebra*. Addison-Wesley, Reading MA.

Garfinkel, R. 1970. Optimal Political Districting by Implicit Enumeration Techniques. *Management Science* 16(8): 495-508.

Glover, Fred. 1968. Integer Programming Over a Finite Additive Group. *AMM-4*, School of Business, the University of Texas at Austin.

Glover, Fred and M. Laguna. 1997. *Tabu Search*. Kluwer Academic Publishers, Boston MA.

Gomory, R.E. 1963. An Algorithm for Integer Solutions to Linear Programs. In *Recent Advances in Mathematical Programming*. R. Graves, P. Wolfe (eds), McGraw-Hill, New York: 269-302.

Gomory, R.E. 1965. On the relation between integer and non-integer solutions to linear programs. *Proceedings of the National Academy Science* 53(2): 260-265.

Gomory, R.E. 1967. Faces of an integer polyhedron. *Proceedings of the National Academy Science* 57(1): 16-18.

Gomory, R.E. 1969. Some polyhedra related to combinatorial problems. *Journal of Linear Algebra and Its Applications* 2(4): 451-558.

Gomory, R.E., E.L. Johnson. 1973. The Group Problem and Subadditive Functions. In *Mathematical Programming*. T.C. Hu, S.M. Robinson (eds.), Academic Press, New York, 157-184.

Gomory, R.E. and E.L. Johnson. 2003a. An approach to integer programming. *Mathematical Programming* 96(2): 181.

Gomory, R.E. and E.L. Johnson. 2003b. T-space and cutting planes. *Mathematical Programming* 96(2): 341-375.

Gomory, R.E., E.L. Johnson, and L. Evans. 2003. Corner polyhedra and their connection with cutting planes. *Mathematical Programming* 96(2): 321-339.

Goodman, Frederick M. 2003. *Algebra: Abstract and Concrete - Stressing Symmetry*. Prentice Hall, Upper Saddle River, NJ.

Grover, Lov K. 1992. Local search and the local structure of NP-complete problems. *Operations Research Letters* 12: 235-243.

Grossman, T., A. Wool. 1997. Computational Experience with Approximation Algorithms for the Set Covering Problem. *European Journal of Operational Research* 101: 81-92.

Herstein, I.N. 1975. *Topics In Algebra*. Xerox College Publishing, Waltham MA.

ILOG. 2003. *ILOG CPLEX 9.0 User's Manual*. ILOG, Mountain View, CA.

Jensen, Paul and J. Bard. 2003. *Operations Research Models and Methods*. Wiley & Sons Inc, Hoboken, NJ.

Johnson, D.S. 1974. Approximation Algorithms for Combinatorial Problems. *Journal of Computer and System Sciences* 9: 256-278.

Johnson, Ellis L. 1980. *Integer Programming--Facets, Subadditivity and Duality for Group and Semi-Group Problems*. SIAM Publications, Philadelphia, PA.

Petersen, C.C. 1969. Computational experience with variants of the Balas algorithm applied to the selection of R&D projects. *Management Science* 13(9): 736-750.

Reeves, Colin R. (Ed.). 1995. *Modern Heuristic Techniques for Combinatorial Optimization Problems*. McGraw-Hill Book Company Europe.

Salkin, Harvey M. and K. Mathur. 1989. *Foundations of Integer Programming*. Elsevier Science Publishers, Salem, MA.

Shapiro, J. 1968a. Dynamic Programming Algorithms for the Integer Programming Problem-I: The Integer Programming Problem Viewed as a Knapsack Type Problem. *Operations Research* 16(1): 103-121.

Shapiro, J. 1968b. Group Theoretic Algorithms for the Integer Programming Problem-II: Extension to a General Algorithm. *Operations Research* 16(5): 928-947.

Stadler, P. F. 1996. Landscapes and their correlation functions. *Journal of Mathematical Chemistry* 20: 1-45.

Toregas, C., R. Swain, C. Revelle and L. Bergman. 1971. The Location of Emergency Service Facilities. *Operations Research* 19: 1363-1373.

Wiley, Victor. 2001. *The Aerial Fleet Refueling Problem.* Ph.D. Dissertation, The University of Texas at Austin.

Weinberger, E. 1990. Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference. *Biological Cybernetics* 63: 325-336.

Woodruff, D.L., E. Zemel. 1993. Hashing Vectors for Tabu Search. *Annals of Operations Research* 41: 123-137.

Wolsey, L. 1971. Extensions of the Group Theoretic Approach to Integer Programming. *Management Science* 18(1): 74-83.

Wosley, Laurence A. 1998. *Integer Programming.* John Wiley & Sons, Inc., New York NY.

Vita

Gary W. Kinney Jr. was born in Sewickley, Pennsylvania on February 28th 1967 to Gary W. Kinney Sr. and Ruth B. Kinney. After completing his work at Fort Cherry High School, McDonald, Pennsylvania in 1985, he enlisted in the United States Air Force. While stationed in Omaha, Nebraska and Kaiserslauten, Germany, he completed undergraduate course work through the University of Maryland and the University of Nebraska at Omaha. He received the degree of Bachelor of General Studies in Computer Science from the University of Nebraska at Omaha in 1995. Upon receiving his degree, he was accepted into Air Force Officer Training School where he received his commission. He attended the Air Force Institute of Technology in 1999 and 2000 where he received the degree of Master of Science in Operations Research. He entered the Graduate School of the University of Texas at Austin in September 2002.

Permanent Address: 646 Bailey's Trail, Dayton, Ohio 45440

This dissertation was typed by the author.